"Intelligent" Internet Search Engine

(Report II)

A report submitted in partial fulfilment of the
requirement for the degree of Bachelor of Science
in Computer Science

by Mr **Antony Richard Howat** (**960776751**)
Department of Information Systems and Computing,

Brunel University

May 2000

Abstract

This paper will consider the design and development of an “Intelligent” Internet search engine, based on research has focused on existing web search technology and the fields of Information Retrieval, document analysis, Data-basing methods and algorithms.

Acknowledgements

Thanks to: Dr. Simon Taylor at Brunel University for his guidance and encouragement as my project supervisor. Olly Betts at Muscat for his help in pointing me at some initial reading into search engine technology. Michael Williams at ARM Ltd for jogging my memory of vector mathematics. Ben Summers at Fluffy Clouds Ltd for further mathematical help. Pace Micro Technology for allowing me to research at work between take-overs and project cancellations. Dr. C. Lee Giles and Dr. Steve Lawrence at the NEC Research Institute for supplying me with reprints of his magazine articles as well as unpublished work. Finally thanks to my mother and father for paying for my continued existence at university. No thanks whatsoever to Microsoft for Word 2000, which caused me many hours of pain and distress during the production of this document.

I certify that the work presented here, and all associated software is my own work unless referenced otherwise.

Signature……………………………………………… Date……………………………

TOTAL NUMBER OF WORDS: 11,567

CONTENTS

CHAPTER 1- SOFTWARE SYSTEM SPECIFICATION

1.0 Problem Definition

The World Wide Web is a sprawling mass of disparate documents in no common format covering all imaginable topics. By its very nature the web is chaotic. Search engines try to make sense of this chaos by presenting a common interface to an indexed version of all these documents. Searching, usually by keywords, will present a list of pages which are possibly relevant to your intended search area.

This project aims to develop parts of an "intelligent" search engine. Most search engines, such as AltaVista, work by storing entire copies of web documents locally in a database and searching them. An "intelligent" search engine does not do this - it downloads documents and only stores the most important parts of a document for index.

Only a human can truly understand and summarise a document, and so have to compromise and use statistical methods developed in the field of information retrieval to get an approximate "jist" of the document. We can also eliminate "useless" search terms, words which have more to do the structure of English than having any meaning in their own right.

There are several justifications for the use of this sort of selective index. In February 1999 it was estimated the most comprehensive search engine on the web only covered 16% of the estimated 800 million publicly accessible web pages. (Lawrence, Lee Giles 1999) Not only is this figure small, but it is decreasing, as the web is growing faster than even the search engines coverage. In a 1997 study the most comprehensive engine covered 34% of the estimated 320 million indexable pages. (Lawrence, Lee Giles, 1999). From these figures it is evident that the existing engines are not keeping up. We may well reach a stage where full text engines cannot index an effective amount of data and still work quickly

Storing just significant parts of documents will reduce the amount of storage needed, and because of this reduction in data volume the search servers with an intelligent search engine can be much smaller and faster in operation - instead of a network of servers searching terabytes of complete documents, one or maybe two desktop machines can comfortably manage to search synopsises of several million documents. Consider the average web page size in an index of 100,000 documents to be 10k. A traditional engine would have to store a full gigabyte of straight pages. If the analysis techniques can reduce each page to a 1Kb index entry the index will only be 100Mb. Search engines typically index many millions of pages, and as more pages are indexed the cost/size benefits become more and more evident.

The faster a search engine runs the more it can feasibly index, and so "intelligent" indexing would go some way to solving the coverage problems of existing full-text search engines, whilst also allowing smaller indexes to run on less sophisticated and less expensive hardware.

1.1 Objectives

- Develop parts of a "intelligent" search engine, enough to demonstrate the techniques involved.
- Consider different architectures for such an engine.
- Investigate and evaluate document analysis techniques.

- Gain a deeper understanding of the concepts involved in information retrieval, and database structuring, and use a combination of these techniques to implement the engine.
- Keep implementation as lean as possible, in keeping with the intention that this engine should be fast and efficient.

1.2 Requirements Specification

The search engine, or parts of a search engine to be implemented must provide basic search engine functionality, in that it will return a ranked list of page matches to a query. It will be a (optionally) stemming, non-full text engine with no concept of word ordering. The documents will be indexed by keywords obtained through automatic document analysis. Emphasis should be on search speed and database size.

As a prototype it is not expected that all the features of a full search engine will be implemented, but enough will be implemented to demonstrate and prove the techniques. The search engine will only analyse and index basic HTML documents, in the English language, but as far as possible the design should allow for future expansion to a full system.

Many algorithms and methods were considered during the production of report 1, many were discounted. Only some of the algorithms and methods discussed will be used in the implemented code.

1) The concept of weightings will be used together with the normalisation methods described in section 3.2.1 of Report 1. A final normalisation method will be decided upon my experimentation.

2) Zipf's Law will not be used but borne in mind, as it was proved ineffectual in the proposed application in section 3.2.2 of Report 1.
3) Document surrogates (section 3.2.3, Report 1) will be both maintained, extracted and used by the engine, but the Dublin Core will not be implemented because of its low uptake.
4) A stop word dictionary (section 3.2.4, Report 1) will be used.
5) Controlled vocabularies and phrase dictionaries (sections 3.2.5 - 6, Report 1) will not be used as their use was not deemed practical for this application.
6) In the interests of speed no compression method (section 3.2.7, Report 1) other than the identification of key terms will be used.
7) Stemming algorithms (section 3.2.8, Report 1) will be used, but implemented as a compile-time option to allow experimentation.
8) SoundEx and thesauri (sections 3.2.9 - 10, Report 1) will not be implemented as they were judged to introduce too much potential inaccuracy.
9) A query method will be implemented depending on time constrains.
10) Vector based matching (section 3.4.3, Report 1) will be used to rank search result relevance. Cosine measure will not be used.
11) String comparison (section 4.1, Report 1) will be performed by standard C library functions, or when used as binary tree (section 4.5, Report 1) by comparing hashes of strings (section 4.2, Report 1). A hashing method will be chosen by experimentation.
12) Inverted indexes (section 4.4, Report 1) will not be used as they were deemed impractical.
13) Chunked file structures (section 4.6, Report 1) will be used intensively in order to create and maintain the database.

Other methods, such as Boyer-Moore-Horspool-Sunday matching will be used, perhaps partially, when deemed relevant.

Such decisions would normally be made at the design stage, however I am specifying the search engine in terms of technical points as it is intended to be a test-bed for the techniques I will use. As such a blanket specification for "a search engine" is irrelevant, as I have to implement a very specific type of engine in order to achieve my objectives. Data flows and the basic modular structure of the system are known already from the research.

1.3 Overall Data Flow

The search engine consists of several distinct modules, as has already been discussed in Report 1, and so the basic data flow is pretty well designed already for us. I stated the modules were :

- A robot to fetch documents from the web, follow links, and feed them to...
- The analyser to parse the HTML, pull out the best terms and feed them to...
- The database - a custom affair with data structures designed to search these lists of words.
- The searcher - parses search terms, ranks results in order of relevance.
- The front-end - providing a simple web interface for searches and submissions, to be fed to the search program or the robot.

Note that in this data flow diagram the searcher and front end have been replaced by a "CGI interface" which performs both their tasks. I decided there was no real need for them to be treated as distinct programs.

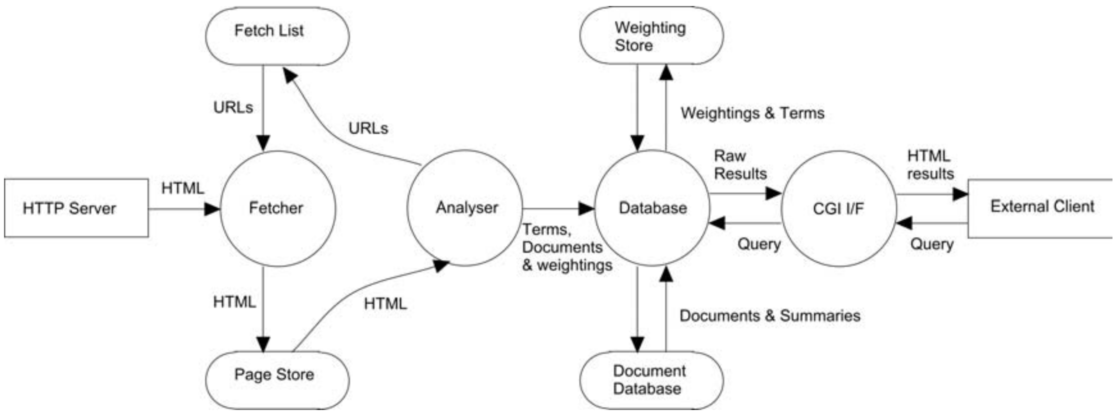


Figure 1: Data Flow Diagram

As well as the design of each logical block on the diagram such things as inter-process communication, and the storage of the data needs to be considered.

CHAPTER 2 – PROJECT MANAGEMENT & DESIGN METHODOLOGY

2.0 Introduction

This chapter details the project structure used to develop the code.

2.1 Development Process Model

The design and development methods used can be considered a hybrid of Rapid Application Development and Structured Systems Analysis and Design. Use of either method in a pure form would be foolish as both have their disadvantages. SSADM, for example, has been widely used by the UK Government, an organisation which is renowned for the repeated failure of its IT implementation projects (Yeates & Cadle 1996, BBC News Online March 2000). The "standard" project management methods are also unsuited to real-world software engineering, and largely act as a palliative for management (Paul 1994). This is an engineering project, not an exercise in management. Rather than crowbarring the project into an unsuitable model, I combined my knowledge of modelling methods into a suitable hybrid.

I separated the project into distinct stages :

1. Inception. The initial project proposal.
2. Feasibility and research. This stage was, in essence, report 1. The feasibility and research sections would typically be isolated in project processes, but they needed to be combined. The research needed to consider many approaches, whilst also considering their feasibility and/or usefulness in the project.
3. Requirements specification. Unlike a traditional requirements specification I also specify how the engine is to work. This is because the whole point of the project was to prove the conclusions drawn in research. A generic search engine requirements specification (just so the design can happen to come up with the same conclusions as the first report) would be superfluous.
4. Design. The design takes the form of separating the task into work units and specifying how they will work in terms of inputs, outputs, interfaces and any caveats concerning implementation. Much of the project, by its very nature, had already been divided into logical work packages, and design decisions had to be made at an early stage due to the insistence on

the presence of conclusions in Report I.

5. Modular implementation. Each module is implemented and that implementation documented through notes and source comments.
6. Modular testing. Each module is tested individually as it is completed.
7. Module integration. The combination of the modules into the complete software package.
8. Overall testing (covered in the following section).
9. Evaluation (covered in the following section).

The end result is something like a double ‘b’ model with concurrency.

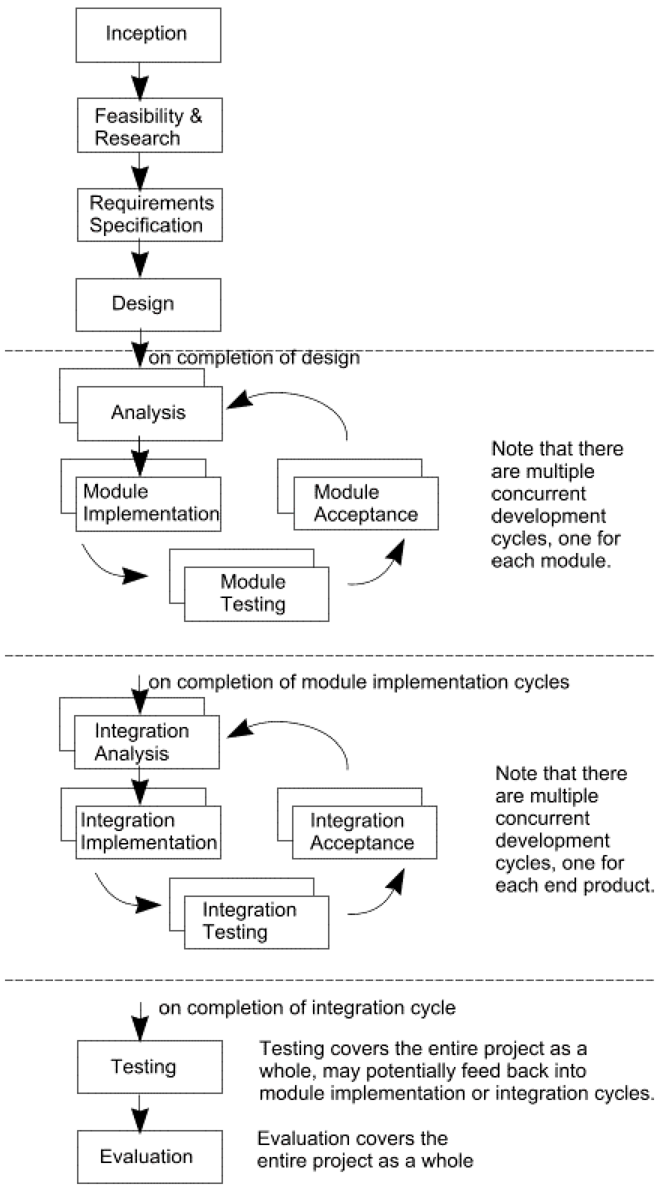

Inception
Feasibility & Research
Requirements Specification
Design
on completion of design
Analysis
Module Implementation
Module Acceptance
Module Testing
Note that there are multiple concurrent development cycles, one for each module.
on completion of module implementation cycles
Integration Analysis
Integration Implementation
Integration Acceptance
Integration Testing
Note that there are multiple concurrent development cycles, one for each end product.
on completion of integration cycle
Testing
Testing covers the entire project as a whole, may potentially feed back into module implementation or integration cycles.
Evaluation
Evaluation covers the entire project as a whole

Figure 2: The process model

The model was treated as flexible. Certain design decisions had to be made out of step with the process, for example the choice of hardware and software tools was made in the first stages of the project so any implementation issues could be put into context.

I imposed no firm time limits on any stage or module, preferring to make sure that modules were implemented reliably rather than quickly. I saw no sense in rushing to meet deadlines produced by estimation. By starting the implementation early this approach worked effectively.

The one deadline I did enforce was a code-freeze a week before submission to allow the write up to catch up, and to focus my mind on documentation.

There is no sense in performing such tasks as gathering project metrics as there is no customer to charge, and there will be no similar follow up projects with which to use the results.

Many software project management methods are concerned with maintaining team communication and motivation. Having spent the past eight years largely managing my own projects I could accurately judge my own potential through previous experience, and as a good end-result was in my interest I could also depend on my motivation.

I could have followed a full project management structure, complete with resource planning, charts and suchlike, and subsequently produced software of very little worth in the remaining time. I would have thus failed to achieve my stated aims. I feel by the fact that I have produced working software in the allotted time proves I used the right management methods for the job, after all management methods are supposed to service the implementation, rather than being a product in their own right. No one ever made money selling project and resourcing plans for toasters to the consumer, they had to implement the toaster.

The entire project can be considered the first cycle of a Rapid Application Development approach. Some modules should be considered “prototype”, and so can be improved and, if necessary, replaced entirely in later iterations. However, unlike a typical RAD project the aim is to prove a set of concepts rather than to deliver an end system for a specific customer. In this respect the user consultation and involvement aspects of RAD were not necessary.

2.2 Testing Strategy

The testing strategy, as described earlier, is split into testing of individual modules as they are developed and the testing of these modules as they are integrated into working programs. As well as detecting bugs the testing fed back into the development cycle, for example if a piece of code ran too slowly it would be re-thought and re-implemented. The overall system testing was performed using a large number of HTML pages stored locally on disc. These pages were from varied sites to give a representative sample. The local storage allowed the database to be re-built quickly without unnecessary network traffic.

All programs during their development were run through a veneer to ensure memory over-runs, under-runs and leaks do not occur. Programs, where possible feed diagnostics to a file or stream to aid debugging.

Given that the end-product is a prototype such things as load testing and testing across a site would be desirable but impractical. If the system were to be productised then I would introduce a comprehensive programme of on and off-site testing with a full database, together with automated stress testing and such.

2.3 Evaluation Strategy

In Chapter 5 of Report 1 I detailed the standard measures of information retrieval performance. These will form the basis of my evaluation strategy, along with the measures of database performance also detailed in the chapter. I will not use van Rijsbergen’s combined measure of information retrieval performance because of its use of intangible measures. However I will measure recall, precision and database speed. Again, all the caveats I predicted in Report 1 apply now.

CHAPTER 3 – DESIGN DECISIONS

3.0 Introduction

In this section we will cover more specific design decisions made relating to each module and the issues identified in the last section. Considering the specific nature of research there is little traditional “pie in the sky” design work to do. The design is more a precursor to the implementation, a rough explanation of what needs to be done and how it should be done. Some areas of code are loosely detailed, leaving decisions which should be straightforward in implementation to the programmer. Others are specified quite strictly, as other code will depend on them. This applies particularly to commonly used APIs and data structures.

3.1.0 Initial Considerations & Approaches

This subsection covers the decisions made before “real” design began.

3.1.1 Platform Selection

It was decided early on in the project that the software would be implemented in ANSI C under UNIX. Whilst the choice of language and OS is more of an implementation issue than one of design, it does influence design. There would be no point in producing a full object orientated design and then deciding that the implementation language would be standard C. The choice of language is based on the low-level nature of the project - I need a high level of control on the data structures used in the program. The use of a higher level language, or many OO languages would have resulted in the loss of control over data formats. For example Java provides a good interface for remote procedure calls, but with its automatic serialisation etc, I cannot be sure that things are being done efficiently on my behalf. C is also, unlike Java, compiled. This is essential for an intensive application, a byte-code interpreter, even with a "Just In Time" (JIT) compiler cannot really compare. C is also highly portable, although we will have to use some UNIX specific operations.

The choice of UNIX as operating system is based on UNIX still being the most popular server platform on the internet, as well as it being the only operating system with enough capacity in terms of scalability to be seriously considered. Windows NT, for example, does not scale efficiently to many processors, and its reliability (even by Microsoft's own admission) is more than questionable :

> *"[Bill] Gates was cited in a Microsoft press release as saying that 'independent testing' by Ziff-Davis Labs showed that Windows 2000 had run for 90 'workdays' without a reboot, compared with just 2.1 days for Windows 95 and 5.2 days for Windows NT 4.0."*
>
> (Lea, 2000)

By comparison a typical UNIX system (commonly Solaris in a server application) will run for many hundreds of days under very heavy loads.

3.1.2 Code Philosophy

The code design would attempt to maximise code re-use, both within this project and in future projects by use of a modular design. For example, the code to implement "chunked" files is written as a generic library, with no project specific code. This approach helps split development into logical units and allows those units to be tested thoroughly before integration.

Where suitable, legally useable, modules of code could be found which would perform a required task it will be used assuming the code is well written, proven to be stable, and not of a prohibitive size.

Code will be commented to a reasonable level, assuming a good software engineer will perform any future maintenance.

3.2 Document Analyser

The document analyser accepts a HTML input, and parses it, extracting text suitable for analysis. This does not necessitate a full blown HTML parser, and can be kept to a reasonably simple. However we cannot be so naïve as to strip all tags from the code, as the HTML includes useful tags which will help in sensibly indexing pages. The major issues in dealing with the initial input before analysing text are :

1. Robustness - The nature of the world wide web means many pages are badly written and include invalid code. This means any code dealing with HTML pages should be robust enough to deal with such pages sensibly without crashing or feeding invalid data to later stages of the system.

2. Use of surrogates - Any useful document surrogates must be extracted and treated with higher weighting than the document body text. An abstract, if one is present, must be stored for each document to be displayed in any results - if no abstract is specifically given it must be extracted from the start of the body text.
3. Preservation of text attributes - Any HTML attributes which text may have which may affect the potential weighting at index should be preserved
4. Recognition of links - The code must recognise and use links within a document to facilitate recursive fetching, i.e. to implement a web crawler. This point includes the recognition of frame links so frame text is also indexed.
5. Interpretation of HTML entities - The text will include encoded entities, for such things as international characters, non-breaking spaces, etc. These should all be dealt with sensibly, in a manner which will not damage the database, but also allow sensible searches.

The code suddenly looks not so simple, considering we have not even considered the processes involved in actual analysis.

The actual analysis will be performed by a function, which is passed a pointer and length of a block of text, along with a structure to give the current HTML state. This block of text will be any constant block of text identified within the HTML. This has implications that words with tags within them (i.e. sp<i>a</i>nner) will not be indexed correctly, but the frequency of such occurrences is negligible. It will then call a function to process any entities. The text analysis function, which now has an unfettered block of text can extract each individual word, and pass it to a function to add each word to a table of occurrences.

The adding a word involves including it on a binary tree structure. This is not the search engine index, but a temporary table of all the words within the document, the number of times they have occurred in the document, along with their hash which is the index value. The structure must be a tree, as sequentially searching a list of even just the words of one document is slow, especially using full string comparison rather than just hashes.

Once all the words in the document have been added to the tree the true analysis can begin. The table should be sorted into order of occurrence (or occurrence affected by a weighting factor according to the HTML tag state of each word) and a table of words deemed relevant along with their individual weightings outputted to the database system for indexing, along with the abstract, word count, source URL etc. The method of calculating normalised weightings will depend on my experiments. The data can then be passed to the database for addition to the index.

3.3 URL Processing

Any links which need to be followed must be added to the robot's list of documents to search. However, again, this process is not so simple. URLs (uniform resource locators) can be relative or absolute, rather like path names in a hierarchical file system. For example, a link from "http://www.foobar.com/pictures/dog.html" to "http://www.foobar.com/pictures/cat.html" could be specified as absolute, or as simply "cat.html". This needs to be dealt with sensibly with respect to the source URL. There are further issues in that, again, like a hierarchical file system back-referencing using ".." can allow a path to work its way back up the hierarchy. From the "dog.html" page a link to "../home.html" would resolve to "http://www.foobar.com/home.html" This back-referencing is not restricted to relative URLs, a link to "http://www.foobar.com/some/other/place/../../../home.html" would be a legitimate (if rather silly) way of referencing the same page. There are also problems with trailing /'s, and the fact that many host names may point to the same host. We need to deal with URLs to carefully to avoid these problems, firstly to deal with relative URLs correctly, and secondly to prevent the same page being indexed many times due to subtle differences in the URL. We do this by processing the URL into its lowest form - removing all back referencing and other such nasties. In other words "http://123.222.222.1/home.html" where 123.222.222.1 is the IP address of the web server. The IP address is ugly, but on presentation to the user it can be processed back into a host name.

To deal with this URL manipulation we need a URL library which provides the basic functionality, namely to disassemble and reassemble URLs. The specification for this is largely provided by RFC1808 (Fielding, 1995), which details the stages involved.

3.4 Inter-Process Communication

Inter-process communication is essential in this project, as the database needs to service both search processes and document analysers. This can be achieved in several ways under UNIX. As one of our aims for the system is that it should eventually be distributable, it makes sense that we only consider IPC methods which will function over a network - there is little impact on speed by using these same methods on a loop-back interface.

Sun RPC (Remote Procedure Calls) is a system which will allow procedure calls between different processes on different machines. This is achieved by a pre-processor program which generates code stubs for the client and server ends, as well as creating serialised data structures to transfer. Structures and exported functions are defined in a simple script language. However RPC does raise some issues, in that it generates code for you, so when a problem arises debugging is more difficult - especially when you consider the clever things RPC will be trying to achieve with possibly complex data structures.

The sensible solution would seem to be to use a simple IP sockets based interface, working with abstract blocks of data. This has some implications :

- Callers have to serialise their own data before transfer and disassemble it afterwards.
- No procedure-call like interface is provided
- Number formats are not fixed. This can cause problems when different operating systems are expected to communicate together. The typical example is a little endian and big endian system communicating.

None of the above implications will cause serious problems if they are kept in mind. Number formats can be converted before transfer in the serialisation stage (typically using BSD sockets' htonl and ntohl routines for converting host byte-ordered numbers to network byte-ordered numbers and vice versa (Richard Stevens, 1990)), veneers can be written to provide procedure call like interfaces for remote functions. Manual serialisation is a pain, but it does give full control of the structures transferred - there is a danger when using automated serialisation methods that far more data than is needed is transferred

Using IP we have the option of a connection-based protocol (TCP) or a connectionless messaging system (UDP). The functionality needed could be implemented with either, but as a call will typically need to return data, possibly a large block of data, it seems sensible to use TCP, especially as a TCP stream can be treated as ‘reliable’ as opposed to UDP which does not guarantee delivery. It would make sense to use TCP streams to create a reliable method of inter-process communication

So a the “Simple Network Interface” was developed. It can be simply replaced with RPC or some similar method at a later date, but is sufficient for current requirements. The protocol definition in the appendicies gives full details on its operation. As we transfer whole blocks of data to perform atomic operations the server design is stateless.

Note that the basic transfer unit for weightings and document details from document analysers is a generic PreProcDoc structure. This design not only gives us a transfer mechanism but allows us to potentially have other analysers for different data formats also submitting to the same database through the same interface, i.e.

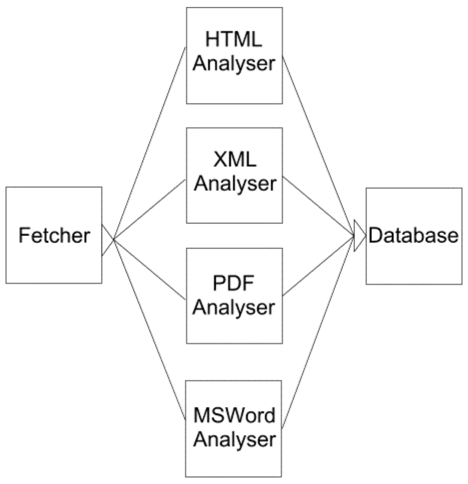


Figure 3: Multiple Analysers Communicating with Database

3.5 Chunked File System

The process of producing chunked files was covered in report 1. They are particularly useful in storing variable length records, which is an essential task in any sort of document index. As such the chunked file interface is as generic as possible, so it can be used for any data type, and many such files can be used at once, with the actual chunk based nature of the files as transparent as possible. The API is as simple as possible. The chunk size for a file is passed when it is opened, and the chunk size may be different for each file that is open - this helps avoid wastage.

File format of a "chunked" file :

```
4 bytes ID String        : CHNK
4 bytes First Free Chunk : offsets from next word, ie first chunk is
                           zero. Chunk offsets are in terms of
                              chunks, rather than bytes.
```

and the chunks are :

```
4 bytes length           : ie, length of data in *this* chunk
                            (-1 = empty)
4 bytes next             : chunk offset to next chunk in this chain
                           (-1 = end of chunk chain)
```

..each chunk followed by a block of data.

The API takes the form of simple C procedure calls.

```
/* open a chunked file (or create a new one if none exists */
chunkfile *chunk_open(char *file,int chunksize)

/* close a chunked file */
void chunk_close(chunkfile *cf)

/* allocate a new chunk chain, and fill it with data */
chunk_id chunk_allocate(chunkfile *cf, char *data, int length)

/* deallocate a chunk chain */
int chunk_deallocate(chunkfile *cf, chunk_id cid)

/* returns a malloced block and length for the data in a chunk chain.
 * user passed pointers to variables to hold data pointer and length,
 * along with chunkfile handle and start chunk_id for the chain.
 *
 * returns non-zero in case of a problem */

int chunk_retrieve(chunkfile *cf, chunk_id cid, char **data, int
                   *length)

/* appends data to an existing chain of chunks, returns 0 if
 * successful non-zero if there was a problem...
 */
int chunk_append(chunkfile *cf, chunk_id cid, char *data, int length)
```

Note that chunk_append is especially suited to a indexing application as it will efficiently add terms and weighting pairs to lists etc.

I would also intend to eventually add functions for modifying a chunk chain, as well as only loading part of a chunk chain.

3.6 Binary Tree Indexed Chunked Files

Later in the design, after specifying the chunks system I realised that an index method would be needed for DocumentRecord and WordRecord files, to facilitate swift access to specific chunk chains. I decided this would be best implemented as a near-transparent extension to the chunks system, which provides a similar interface to the chunks code but also automatically maintains a binary tree index of the records. This index could be stored in the actual chunks, or separately. I soon realised that storing the index in the chunks would be nightmarish by simply thinking ahead to the implementation (load a chunk with its header, skip the header and pass it back to the user as the data, the pointer is then invalid and cannot be freed by the user, so either the code has to copy the data and then pass the copy (slow), or provide a free() veneer to release the memory correctly. There are similarly ugly issues when it comes to scanning the index etc). Instead the code should maintain a separate index file containing the tree.

Again, as the code would be used throughout the implementation the API was specified early.

```
/* open a chunked btree file
 */
cbtfile *cbtree_open(char *file, int chunksize);

/* close a chunked btree file
 */
void cbtree_close(cbtfile *cf);

/* find the number of individual chunk chains in a file */
int cbtree_countentries(cbtfile *cf);

/* just locate a chunk in cf (returns an integer index for cbindex
 * record for that hash) - returns -1 if no such entry in the index
 * exists. returned figure should be passed to future operations on
 * this hash - it is an indexpoint.
 */
int cbtree_locate(cbtfile *cf, int hash);

/* retrieve a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns chunk_id of main file containing matching record
 */
chunk_id cbtree_retrieve(cbtfile *cf, int hash, char **data, int *length, int
ichnk);

/* append to a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns 0 if successful,
 * non-zero if there was a problem...
 */
int cbtree_append(cbtfile *cf, int hash, char *data, int length, int ichnk);

/* get chunk file handle */
chunkfile *cbtree_getchunkfilehandle(cbtfile *cf);

/* add a new hash to the index and create chunk
 */
chunk_id cbtree_allocate(cbtfile *cf, int hash, char *data, int length);

/* returns chunk pointer for a cbindex entry
 */
chunk_id cbtree_getchunk(cbtfile *cf, int index);
```

The API includes indexpoints, which as the name suggests are pointers to the element in the index for that record. This feature can be used to save multiple traversals through the binary tree when several operations are done sequentially to a single chunk_chain.

3.7 Database Design

The database makes extensive use of the chunked file libraries to allow the storage of word lists and document information. This means we can safely design data structures which are variable length to form records in the various files.

Due to the custom and low level nature of this database the traditional methods are not strictly appropriate or relevant. I have approached it from a lower level, and specified it in terms of data types and structures.

The basic design is as follows :

```
URLRecord
   char URL;

DocumentRecord
   int urlhash;    // hash of lowest-form url
   chunk_id left;  // Binary tree pointers to other DocumentRecords
   chunk_id right;    // keyed by the hash of their URL
   int sourceurl;     // points to a chunk in a urls file
   char title;        // fixed length space for document title
   char abstract;     // fixed length space for abstract
   time tm;           // time of document index (for future use,
                    // expiry etc)
  int words[]...    // list of word records follows, a WordRecord
                    // reference for every document this word appears
                    // in, so we can eventually delete the references
                    // without searching every term we have indexed
WordRecord
  char word;         // text representation of word
  int hashword;      // hash of word
  chunk_id left;       // binary tree pointers to other WordRecords...
  chunk_id right;    // keyed by hash
  docweight wlist[]... // list of document and weighting pairs,
                     // one entry for each document containing word
```

The docweight type is simply a grouped DocumentRecord chunk_id and weighting value pair. Chunk_id can be assumed to be a 32 bit integer.

The URLs are stored in their own chunked file as the length of URLs is not restricted and varies wildly from site to site. A large fixed length field would be wasteful, and so I intend to use a small chunked file.

To illustrate the structure, consider a document at a URL "http:/www.test.org/thing.html" containing the index terms "Mr" and "Lunch" :

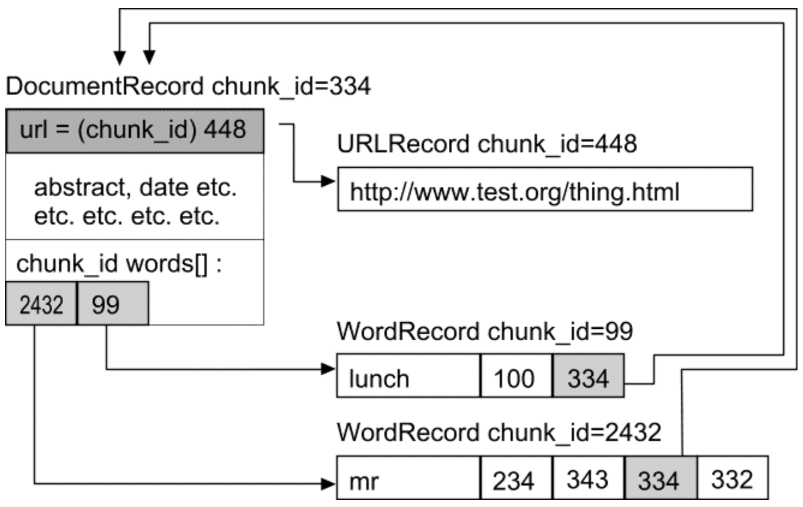


Figure 4 : Simplified database diagram showing references between tables

The diagram does not take into account the tree indexing of WordRecord and DocumentRecord structures for simplicity.

To prove the structures I dry-ran through the common tasks of a search engine, namely adding a new document to the index, performing a word search, deleting a document, and checking the presence of a URL.

Performing a word search

1) Hash ASCII value of stemmed word.
2) Follow binary tree of WordRecords using hash as an index.
3) Load chunk with same hash.
4) Compare ASCII values (to ensure no words with the same hash get confused). If they do not match continue through the tree.
5) If ASCII values match grab list of documents and weightings from chunk.

Submitting a document

Having received compiled normalised document statistics in a transfer structure from an analyser a DocumentRecord structure is built and added to the DocumentRecords structure's binary tree. The following is performed for each word :

1) Hash ASCII value of stemmed word.
2) Follow binary tree of WordRecords using hash as an index.
3) Load chunk with same hash.
4) Compare ASCII values (to ensure no words with the same hash get confused). If they do not

match continue through the tree.

5) If ASCII values match add chunk_id of DocumentRecord and weighting for that term to the weighting/document list. If the ASCII values do not match a WordRecord for this term does not exist, so create a new chunk and add to that.
6) Add the WordRecord chunk_id to the words list in the DocumentRecord structure.

Deleting a document

1) Retrieve DocumentRecord structure for this document (indexed by hash of URL).
2) Load chunk, and proceed to remove DocumentRecord chunk_id and associated weighting from every WordRecord chunk_id listed in the words list within DocumentRecord.
3) Delete DocumentRecord structure.

Checking presence of a URL

1) Find base-form URL of document
2) Hash it, and search DocumentRecord file using URL hash as an index.
3) Compare ASCII values of URLs, if they match URL is present, if they do not try to continue through the tree. If this is not possible the URL is not in the index.

Since the design of the Binary Tree Indexed Chunked Files all the tree operations detailed in this section will be dealt with transparently by that module.

3.8 The CGI Interface

The CGI interface will include what is the searcher functionality, by interpreting queries, passing them on to the database using the networking code, interpreting the results returned and formatting them into HTML.

The initial intention was to implement a boolean style query system perhaps with weightings, but having researched the typical use of existing search engines, I decided my time could be spent better elsewhere on the project. Very few people actually use boolean or extended boolean queries, this can be attributed to the extra syntactic complexity and the fact that people just are not that specific when specifying an information need.

It will rank relevance using the adapted vector based matching formula I described in Report I. One major design decision has not been taken, in that I am not committing myself to any particular query form. A boolean query system with parenthesis may be implemented, but in many cases a simple word search with no logic operations is just as effective. A simple multiple word search is easier to implement, and so depending on time constraints I may well limit the interface to that.

3.9 The Robot

The robot will download pages via HTTP, store them temporarily on disc and submit them to the analyser. The addresses for the pages to download are taken from a list, which is constantly added to by the document analyser as it finds links to follow. For this prototype I have decided not to implement a full robot, but to use a UNIX fetcher tool named "wget" , perhaps supported by some UNIX shell scripts to provide the extra functionality - a simple web fetcher may be easy to implement, but there is enough work to be done already. Plus my network location necessitates further complexities such as HTTP authentication, which wget can already cope with. It should also be noted that the prototype will more than likely be processing pages stored on disc rather than actively crawling the web at such an early stage.

CHAPTER 4 - IMPLEMENTATION

4.0 Introduction

This chapter details the implementation of the system.

Constants which define the length of fields or blocks of data in structures are defined as macros to allow them to be adjusted relatively easily with a simple recompilation. This avoids potential problems when field sizes are adjusted. Such attributes could be implemented so they can be configured more readily at a later date.

Note that the timing module, "shavtimer" is not covered in this section, but an appendix as it was implemented at a later stage and exists purely for gathering metrics.

Source File	Description
chunks.c	Library to implement chunked files, storing multiple records of arbitary length
cbtree.c	Extension library which transparently uses chunks.c to provide similar behaviour but indexing all records using a hash-key which it uses to create a B-tree index for quick searches.
server.c	The main data-server code, primarily dealing with network connections and despatching client requests to db.c.
db.c	The code for handling the data within the server, providing all the basic search engine operations as procedures for server.c to call.
preprochtml.c	The main HTML preprocessor, accepts a HTML file and analyses it for key terms, submitting a final ranked table to the database using sni.c.
sni.c	The basic networking interface for clients to communicate with the data server.
tags.c	Library for processing HTML tags, splitting them into their tag bases and individual attributes for processing.
entities.c	Library for processing a block of HTML, replacing all entity strings with the correct character and vice versa.
stopwords.c	Library to load a list of words from a file and then reject or accept words for addition to an index according to whether they are present in the list or not.
porter.c	Code to stem a string according to Porter's stemming algorithm.
fetch.c	Main CGI interface, accepts parameters from users in CGI variables and returns

	HTML to web server containing search engine responses.
util.c	Generic utility functions, for such things as loading a file into an allocated block of memory etc.
debug.c	Generic debugging functions, for such things as hex-dumping a block of memory to an output stream etc.

Table 1 : Table showing major source files in search engine

uri.c is not included in this table as it is not linked into the current version of the project.

Note that although "fetch" may not seem the most logical name for the CGI interface it was chosen as "fetch" had become the working name for the system, as such the CGI interface belonged to "fetch". Another search engine based around exactly the same analysers and database may well have a radically different piece of CGI interface code in order to provide it with a unique interface.

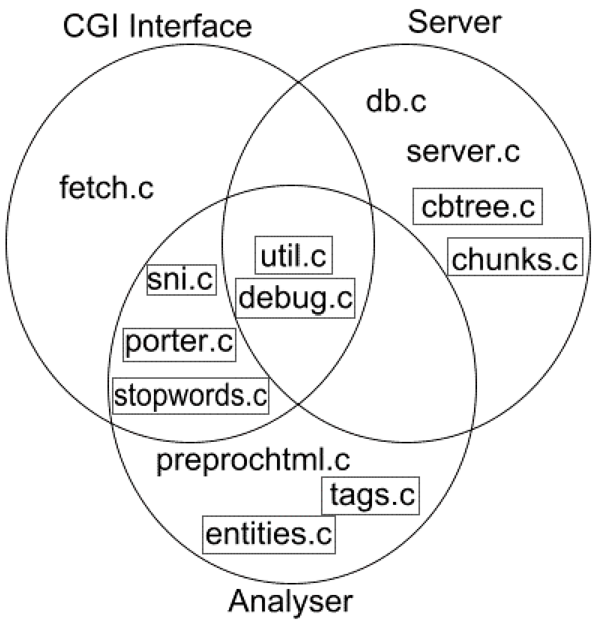


Figure 5 : Venn diagram showing code-sharing between executables

(Boxed source file names are packaged as re-useable modules)

From the diagram we can see all the executables share common utility and debugging functions, other than these the server is pretty well isolated from the other two programs. The server does in fact share data structures with the other two programs, but no actual code. The diagram does not show that although cbtree.c and chunks.c are only linked to the server the are actually re-used within that code, in that they are used multiply to implement the word list, url list and document records. The CGI interface and the analyser share the code used for client side communications to the server, as they both have to talk to the database. They also share the code for processing index terms, porter.c and stopwords.c –

this follows the basic rule of data reduction, in that for an index to work the same methods must be applied to the query terms as are applied to potential index terms.

4.1 The Chunked File System

The chunked file system was implemented as a re-usable library as per API in the specification. Upon opening a file it returns a chunkfile structure pointer which is used as a handle. Within that structure the chunks code maintains all its state information and could potentially store such things as a cache. As such the chunks code can maintain many open files at once independently. This is exactly what is needed as the server stores three separate databases as chunked files.

The implementation of the code is fairly straightforward use of C file operations. A crucial macro is :

```
CID_TO_BYTE(cid,clen) (cid*(clen+8))+sizeof(chunkheader))
```

It calculates the byte offset of a chunk ID, where the chunk data areas are of size *cid*. Reading a chunk chain is simply a matter of seeking to that chunk, reading the chunkheader, reading the data, seeing if there is a following chunk in the chain and if so reading that in too until the chunk chain ends.

Allocating chunk chains is more complex as there can potentially be a free list of chunks to re-allocate. This is pointed to by the free list header in the header of the chunk file. If it is set to –1 the next chunk should occur at the end of the file. The routine then simply allocates and writes chunks, filling them with data until there is no more data left. When a chunk header is being constructed the code determines if there will be any further chunks needed after this one, and if so pre-allocates it. This avoids having to write the "parent" chunk's header twice. The append routine works in the same way, but before it starts writing chunks it needs to find the chunk chain to append to and fill any remaining space left in it.

The free routines, although not yet used in the full program (as no deletion or updates are supported by the engine as yet) works, in that it will de-allocate a chunk chain, adding each item to the free list for future allocation. At present as it follows a chunk chain which is being freed each element is added to the free list head in turn, and so the result is a chain of free chunks that runs backwards. To correct this the free chunks could be buffered by the routine, sorted, and then written out. My preferred solution, however would be to periodically read in the free list, sort it, and then update the links in it. This way not only individual ex-chunk-chains would be sorted, but the whole list would be organised.

Concurrency control is provided by file locking using the standard SysV facilities (Richard Stevens, 1990). The chunk routines at present lock the entire file on any read/write operation. This is to prevent interference between threads of the server. The same SysV functions also supply functionality for record locking which could be implemented in the future. For example when adding to a chunk chain just the chunks involved could be locked rather than the whole file. This would reduce the amount of time other threads would potentially be blocked by one processes operations.

The initial testing of the module was performed by adding large files as chunk chains, deleting them, re-adding them, appending to them, and generally using all the operations intensively, whilst verifying the states. Like much of the code the chunk code has a debug mode which dumps a large number of diagnostics.

4.2 The Binary Tree Indexed Chunked File System

The code for this module was implemented as a layer on top of the chunks code, maintaining a simple binary tree index for the records in a chunks file. The API is kept as similar as possible (see cbtree.h in the source listings), whilst also allowing the caller to supply index points rather than hashes in certain cases to reduce the number of tree searches. This is specifically for situations where a routine may first check the location of a record, and then modifying it – without being able to pass index points the operation would involve an unnecessary pass through a potentially large index.

The hash keys themselves are generated by the caller, and the code assumes a 1:1 mapping between all hashes, in other words that we are using a minimal perfect hashing function. Although we did prove the reliability of the polynominal method of hashing in section 4.2 of report 1, future versions of the code should allow for hash collisions by allowing for hash collisions.

As this API uses the chunks system for its underlying functions it only need worry about concurrency of the index trees which it maintains itself. It does these in the same way as the chunk code at present, using simple file locking.

It was implemented and tested, and found to be working fine. That was until it was integrated into the server, when the tree would occasionally lose entries. This caused about two week’s worth of head-scratching until I found that I had not switched buffering off on the indexes, so the different threads of the server were happily writing over each others additions to the tree. The busier the server, the more losses (presumably buffers are automatically flushed less regularly when the system is busy). This is why there is an almost excessive amount of error checking in the code, reporting such unlikely circumstances as looped references in the tree.

4.3 The Network Interface

The network interface was implemented as per the specification, initially alongside a bare-bones version of the server to test against. The client side API is extremely simple :

```
/* sni_send
 *
 * Send an arbitary block of data to a remote system using a TCP stream,
 * recieve any response (either a response code or a whole block of data)
 *
 * char *dest        destination hostname
 * int  port         IP port number
 * int  type         transfer type
 * void *data        data to send
 * int length        length of data to send
 * sni_response *sr  pointer to sni_response structure to be filled
 *                   with returned data and response code
 *
 * returns : char * containing a simple error message or NULL
 *
 */
```

A sni_response type was defined to package the returned data, its length and type :

```
typedef struct _sni_response {
  int length;
  char *data;
  int type;
} sni_response;
```

This fragment of the test code gives a simple example of how it is used :

```
int main(void)
{
  char test[]="This is a test\n";
  char *err;
  sni_response sr;

  err=sni_send("mipc-21",1234,1,test,strlen(test)+1,&sr);
  if(err)
    printf("%s\n",err);

  printf("sr.type=%i sr.length=%i sr.data=0x%x\n",sr.type,sr.length,sr.data);
  if(sr.data!=0)
  {
    sr.data[sr.length]='\0'; // terminate the string
    printf("%s\n",sr.data);
  }

  return(0);
}
```

Note that in the actual project the port and hostname of the server are #defined macros.
The process effectively blocks sni_send is called, and it returns only when an error condition has occurred or a response has been received. The client itself is implemented using standard UNIX BSD sockets. The code is paranoid about error conditions, and conditions where the server may respond with too much data. If this went unchecked the client would crash with memory over-runs. To avoid this the area is progressively re-allocated as data arrives. If the amount of data exceeds the length specified in the response header the response is rejected and an error is returned.

The implementation of the server end is detailed in the server section, and the protocol definition can be found in appendix B.

4.4 Porter’s Stemming Algorithm

The implementation of Porter’s stemming algorithm used is a reference implementation from the inventor’s homepage (Porter, 1999), with a small wrapper function to make it accessible to the rest of the code. As such it is not documented here.

4.5 The Entity Library

The entity library consists mostly of a table containing the data from the HTML specification detailing the translation of HTML entities, ie “&amp;” to an ampersand etc. This data is stored as an array of EntityEntry types, the type consisting of a string and its ASCII equivalent character. There are two main user functions, EntitiesDeEntitise, which takes a block of text and returns it with all the entities reduced to their ASCII equivalents. This code works directly on the passed data, as the length will always reduce. The code has to allow for bad HTML, such as entities which are not terminated correctly, as well as dealing with numeric specifications. The entity is removed and replaced with the translated ASCII character, and the following memory is moved back to cover the gap where the remainder of the entity name was. This repeats until the string is free of entities. The EntitiesEntitise() function does the reverse, and unlike DeEntitise the output will typically be longer the original, and so it returns a newly allocated block containing the results to avoid any potential over-runs.

The module was initially tested by processing a test HTML file which contained a table of all the entities, dentitising it, and then entitising it again.

4.6 The Tag Parser

The code processes HTML tags in the form :

```
<tag name=”foo” content=”moo” scheme=”baa”>
```

There can be any number of attributes to a tag, and the attribute values can be of any length. The initial call to TagParse takes the tag as a whole (minus the surrounding <> characters), and processes it into a TagData structure, which contains a linked list TagAttribute structures containing their names values.

```
typedef struct _TagAttribute
{
  char *name;         /* an argument in a HTML meta tag */
  char *content;
  struct _TagAttribute *next;
} TagAttribute;

typedef struct
{
  char *name;          /* contents of a HTML meta tag */
  TagAttribute *attributes;
} TagData;
```

The block pointer returned is then used as a handle for TagGetAttribute calls which simply search the linked list for a variable name and return the value if there is one. After using the processed tag a call to TagFree releases all the memory for the data and names in the linked list, as well as freeing the parent structure. Nothing clever is involved in the implementation as it is just a simple utility function.

4.7 The Stop-words Library

The stop-words library is implemented in a very simple manner. stopwords_load causes it to load a list of either comma or white-space separated words from a file on disc into memory. It is loaded as is. The IsValid routine will then scan the list, but using strstr() which is a routine for finding a sub-string within a string. If a match is found, and the character following the match is a comma, a null, or a whitespace character, and the character preceding the match is a comma, space or the start pointer to the file then it has been found in the stop-words file and is reported as such. These checks prevent sub-strings of stop-words being omitted from the database by accident. The closedown routine free_stopwords() simply de-allocates the memory used for the stop-words file.

4.8 The Document Analyser

The document analyser has been implemented with the aim of processing the document from memory in performing as many operations as possible concurrently. The fact it processes from memory allows quick pointer based operations on the data, and also allows for future expansion in that a future version of the fetcher could have no disc buffering, and simply feed fetched documents directly to the parser as blocks of memory.

The main procedure is PreProcessHTMLDocument, which accepts a string containing the HTML document data, a length in bytes and the original URL of the document. The code within PreProcessHTMLDocument is concerned with interpreting the HTML, identifying and dealing with tags that are relevant to us, and passing on chunks of body text to be processed. It allocates a PreProcessedDocument structure :

During the process the program is also attempting to fill in its abstract, either from a given meta-element or from the first few bytes of the body text. This occurs concurrently as the loop searches through the document for potential search terms.

The process for each individual word runs as follows :

- The analyser also a bit-field to identify which characters are upper case in a term, this is redundant in that the data is not considered in ranking. It may prove useful in the future in identifying such things as proper nouns and anacronyms.
- The stopwords list is consulted to see if the word is valid for index or not, if not it is rejected.
- The CRC of the word is calculated, this is used to search a binary tree of potential index terms to see if it has already been encountered.
- A new entry is created in the binary tree for that word, or the existing entry is updated to have its weight increased. Each entry is stored with its upper case bit field, the number of occurrences, the weighting (which is not a full weighting, but a measure of importance which will give a term a greater chance of indexing if, for example, it was found in the title of the document).

The construction of the initial index of potential terms is a potentially slow process. The first implementation used a standard list of strings, allocated in a block. Each index term was compared sequentially using string comparison. Indexing a 250k document on a 280Mhz Sun Ultra 5 was taking over 30 seconds, which was an unacceptable delay. A binary tree was the obvious solution, and as a binary tree index value was needed the CRC value seemed sensible. The use of the CRC as key meant there were no string comparisons involved in the search. The analysis time subsequently dropped to being so short I could not accurately measure it. The code was further optimised by the blocking of memory allocation into chunks of several elements. This way the number of allocations (which each take time) was reduced, further optimising the process. Memory allocation will be particularly slow as we are also extending a single uniform block of mamory, so it may well have to be moved to accommodate its new size. The reason for using a single uniform block will become apparent later. The blocking was simple to achieve, as is illustrated by the actual code :

```
if((ppd->wordlistnext % WORDLIST_GRANULARITY) == 0)
{
  /* we need a new block of memory */
  if(ppd->wordlist)
  {
    ppd->wordlist=realloc(ppd->wordlist,
                        (ppd->wordlistnext+WORDLIST_GRANULARITY)*sizeof(WordRecord));
  } else {
    ppd->wordlist=malloc(sizeof(WordRecord)*WORDLIST_GRANULARITY);
    ppd->wordlistnext=0;
  }
}
```

Once the list of potential index terms has been constructed it needs to be sorted. Sorting a binary tree of individually allocated nodes would involve allocating a suitably sized table and filling in the results from there. By the use of a single block of memory to store the entire tree we avoid this problem, and can use the nifty C library qsort() function to sort the entire list in one fell swoop.

The process involved with selecting qsort() was a simple one, it is quick and part of the standard C library. I would consider other methods if the sort proved to be too slow, but it was not. I would liken it to being given a spade, asked to dig a small hole. You would not spend days building a mechanical digger to do the work more efficiently. Another point is that a qsort() does not necessarily equate to a quick sort, a C library may implement it using a different algorithm and provide the same interface. The code may well also be optimised for the destination platform.

After sorting the left and right pointers of each element become invalid, but they are not needed any more as the searching stage has completed.

Finally SubmitDocument() takes the PreProcessedDocument structure and sends it in a slightly modified PreProcDoc structure (mostly the same but lacking things like pointers which would be irrelevant if transferred) along with a manually serialised list of the top few terms and weightings to the server. They are then sent using the simple network interface client.

4.9 The Data Server

The data server is implemented in two main source files, server.c which deals with the network communication and processing of requests into database requests which are passed on to relevant functions in db.c.

The server code starts the database (by calling db_start from db.c) and then sets up a TCP/IP server socket, listening and forking a process for each connection that is accepted. This is best illustrated by the code itself :

```
/* this routine starts a server process, listening for connections
 * and forking off further processes as connections occur
 */

int main(void)
{
  int ns, fromlen;
  struct sockaddr_in sin, fsin;

  setbuf(stdout,NULL);

  log_event("Search engine data server (" __DATE__ " " __TIME__ ")");

  /* set signal handlers */
  signal(SIGCHLD,SIG_IGN);
  signal(SIGTERM,(void (*)())sig_term);

  log_event("Opening database...");

  if(db_start(DBROOTFILE))
  {
    log_event("Failed to open database at " DBROOTFILE);
    exit(1);
  } else {
    log_event("Database at " DBROOTFILE " opened OK");
  }

  /* Get a socket to work with */
  if((s=socket(AF_INET,SOCK_STREAM,0)) < 0)
  {
    fprintf(stderr,"main: Failed to create socket for server\n");
    db_finish();
    exit(1);
  }

  /* create address to bind to */
  bzero((void *)&sin,sizeof(sin));
  sin.sin_family=AF_INET;
  sin.sin_port=htons(SNIPORT);
  sin.sin_addr.s_addr=htonl(INADDR_ANY);

  log_event("Binding socket...");

  /* bind to socket */
  if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
  {
    fprintf(stderr,"main: Failed to bind server\n");
    db_finish();
    exit(1);
  }

  log_event("Calling listen...");

  /* listen to the socket */
  if(listen(s,5)< 0)
    {
      fprintf(stderr,"main: Failed on accept, errno %i\n",errno);
      exit(1);
    }

    /* fork a child process to deal with new connection */
    if((pid=fork())<0)
    {
      fprintf(stderr,"main: forking process failed\n");
      close(s);
      close(ns);
      exit(1);
    } else {
      /* do child-like things */
      if(pid==0)
      {
        close(s);
        childtask(ns,&fsin);
        exit(0);
      }

      close(ns); /* close parent copy of socket */
    }
  }
  exit(0);
}
```

The use of fork() was a design decision taken at implementation time, it causes a copy of the process to be made to maintain the connection.. The alternative was to have a single process which dealt with all connections, using select() to scan connections for activity. This is marginally more difficult to implement, but does save memory and eliminates potential concurrency problems, by way of eliminating concurrency. I decided it was preferable to use forking as the processes could potentially run in parallel on a multi-processor system. Plus the fact that each session is a separate process improves stability, if one client process crashes it will not kill the entire server, just the process for the offending connection.

The childtask() function then listens to the incoming socket for data in the form we specified earlier as our simple network interface. The packets of data are received in a malloced block which expands as more data arrives. A switch statement then looks at the transfer type to determine which code to run to transform the block of data into the parameters needed for the equivalent function in db.c.

db.c utilises the libraries to provide the database functions, namely searching, document information retrieval and the addition of new documents to the index. The operations are simple, but rather longwinded, and are best illustrated by the source itself and the commentary within it. The code is simply an interface between the database and the operations requested by the client.:

db.c also provides a couple of miscellaneous diagnostic functions, including a call for extracting a HTML formatted table of database statistics and timings to be sent to the client. There is also a compile-time define which causes db.c to save file size statistics periodically to a table – this was used to analyse the efficiency of the database.

4.10 The CGI Interface

The CGI program is run by the host machine’s web server when a search is requested. The search terms are supplied to the program by parameter extensions to the URL. These are parsed into a more useable form by code which was lifted from John Marshall’s “CGI Made Really Easy” (Marshall 1998). This code required some debugging before it was stable under Solaris running Apache.

Queries are passed as a “q” variable in the url, i.e. a query for “monkey nuts” would be done passing the following URL:

```
http://mipc-21/cgi-bin/fetch.cgi?q=monkey+nuts
```

The query terms are then passed through the same stop list and stemming algorithm as all the terms were when they were being indexed. These new stemmed terms are reported to the user in the results, to provide context so the user can optionally improve his terms considering their stemmed results.

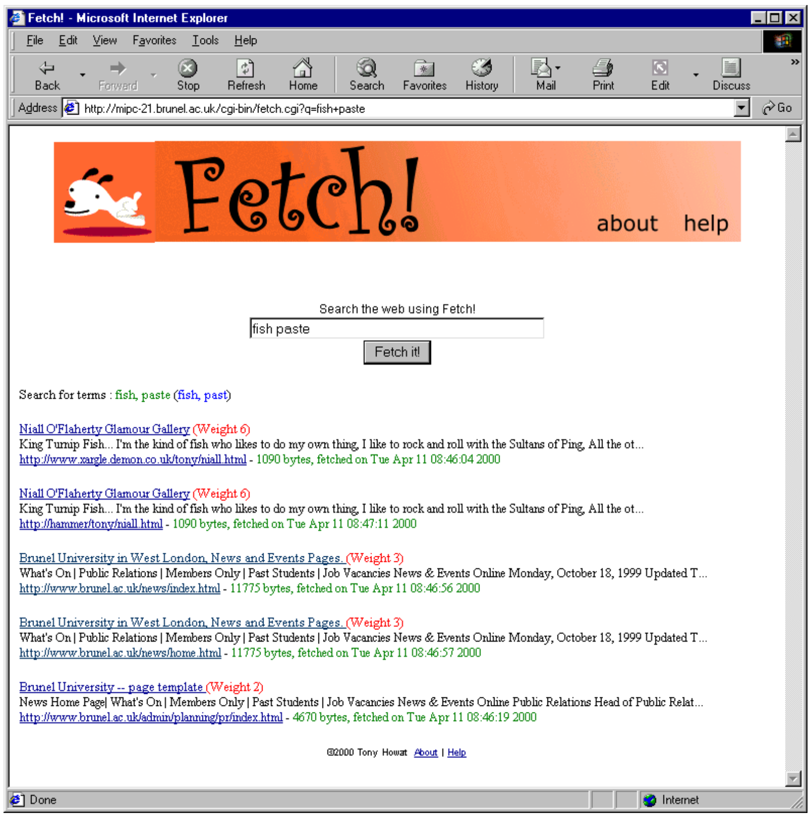
The stemmed terms are sent to the database server in a manually serialised form, in other words a 32 bit integer giving the number of terms, followed by a sequence of fixed length strings of *TERMSTRINGLENGTH* each.

The server then responds with a best-match-first ordered list of document identifiers (hashes of document’s base-form URLs) and relevance ratings. The client then takes as many of these document identifiers as it needs and sends a request for the DocumentRecord structure for each. From this structure each document’s title, abstract and such are obtained. This data is then formatted into HTML and sent to the output stream, ie. the HTTP interface.

There is a second CGI variable that the interface responds to. The presence of a “info” variable (whether empty or not) in a query URL will cause the interface to dump the search engine performance statistics and general information before doing any query.

The CGI interface code is the least pleasant in terms of structure and style, as all it is doing is essentially spewing data to an output stream in a HTMLised form.

The other major aspect of the CGI interface is that it actually implements the human-computer interface. As mentioned before, the interface design for the project is simple, as the major focus of the project is not on user interface issues. As decided at the design stage, the human computer interface is a simple non-boolean query, essentially a list of search words. The actual design is a simple HTML input form, and the output follows that of a typical search engine. Results are returned as a list of pages ordered by their relevance to the query, again as a HTML page. Each result will link to the page it refers to, and each entry in the results list will include the document title, a brief abstract and a rating of relevance, similar to that produced by HotBot and pretty well any other search engine for that matter.


Fetch! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mipc-21.brunel.ac.uk/cgi-bin/fetch.cgi?q=fish+paste
Go
Fetch!
about help
Search the web using Fetch!
fish paste
Fetch it!
Done
Internet

Search for terms : fish, paste (fish, past)

Niall O'Flaherty Glamour Gallery (Weight 6)
King Turnip Fish... I'm the kind of fish who likes to do my own thing, I like to rock and roll with the Sultans of Ping, All the ot...
http://www.xargle.demon.co.uk/tony/niall.html - 1090 bytes, fetched on Tue Apr 11 08:46:04 2000

Niall O'Flaherty Glamour Gallery (Weight 6)
King Turnip Fish... I'm the kind of fish who likes to do my own thing, I like to rock and roll with the Sultans of Ping, All the ot...
http://hammer/tony/niall.html - 1090 bytes, fetched on Tue Apr 11 08:47:11 2000

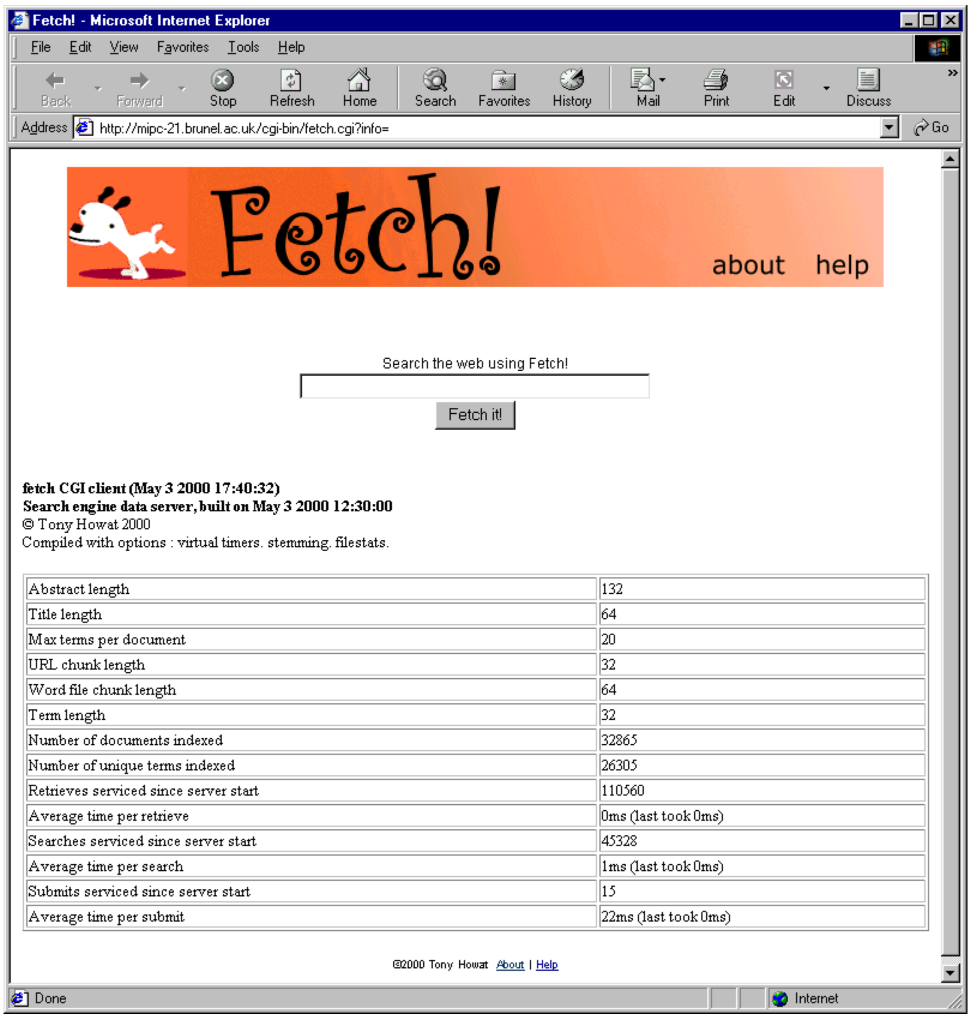
Brunel University in West London, News and Events Pages. (Weight 3)
What's On | Public Relations | Members Only | Past Students | Job Vacancies News & Events Online Monday, October 18, 1999 Updated T...
http://www.brunel.ac.uk/news/index.html - 11775 bytes, fetched on Tue Apr 11 08:46:56 2000

Brunel University in West London, News and Events Pages. (Weight 3)
What's On | Public Relations | Members Only | Past Students | Job Vacancies News & Events Online Monday, October 18, 1999 Updated T...
http://www.brunel.ac.uk/news/home.html - 11775 bytes, fetched on Tue Apr 11 08:46:57 2000

Brunel University -- page template (Weight 2)
News Home Page| What's On | Members Only | Past Students | Job Vacancies News & Events Online Public Relations Head of Public Relat...
http://www.brunel.ac.uk/admin/planning/pr/index.html - 4670 bytes, fetched on Tue Apr 11 08:46:19 2000

©2000 Tony Howat About | Help

Figure 6: The Web User Interface


Fetch! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://mipc-21.brunel.ac.uk/cgi-bin/fetch.cgi?info=
Go
Fetch!
about help
Search the web using Fetch!
Fetch it!
©2000 Tony Howat About | Help
Done
Internet

fetch CGI client (May 3 2000 17:40:32)
Search engine data server, built on May 3 2000 12:30:00
© Tony Howat 2000
Compiled with options : virtual timers. stemming. filestats.

Abstract length	132
Title length	64
Max terms per document	20
URL chunk length	32
Word file chunk length	64
Term length	32
Number of documents indexed	32865
Number of unique terms indexed	26305
Retrieves serviced since server start	110560
Average time per retrieve	0ms (last took 0ms)
Searches serviced since server start	45328
Average time per search	1ms (last took 0ms)
Submits serviced since server start	15
Average time per submit	22ms (last took 0ms)

Figure 7: The information/statistics display

4.11 The URL library

Whilst the URL library is not used in current test versions of the search engine code, it is included as it is a significant part of the project which would come in to play when it is let loose on the internet. It was also one of the most challenging modules, for the reasons described in the design section. RFC1808 (Fielding, 1995) does provide a step by step guide to interpreting URLs and URIs, but this is simply not enough. A straight implementation will not work in many cases, because, as with many things on the world wide web, standards are incredibly flexible.

I initially considered using code from HTMLLib from Mozilla, but found it to be too repulsive and attached to the rest of the library.

The code centres around pulling URLs out into the following structures :

```
/*
 *  These two structures are used to pull a URI apart ie :
 *
 *  http://user@www.moo.org:8080/directory1/directory2/index.html
 *  |      |    |           |                          |
 *  |      |    |          port                       leafname
 *  |      |    host
 *  |      username
 *  transport
 *
 *  Path section is turned into a linked list of URIPathElements.
 *
 */

typedef struct _URIPathElement
{
  char *name;                    /* element of path */
  struct _URIPathElement *next;
} URIPathElement;

typedef struct
{
  char *transport;
  char *user;
  char *host;
  char *port;
  URIPathElement *path;
  char *leafname;
} URIExtruded;
```

The URIExtrude function parses a text URI into its parts, putting them into a URIExtruded structure. The path elements are put into a linked list hung off the URIExtruded structure. It also includes the functionality for combining a base URI with a relative URI – this is because such things as back references ("..", "." etc.) although usually only sensibly included in a relative URI may also appear in a base URI, so both parts need to be processed in the same way. Predictably the URIUnextrude procedure reverses the process, turning an extruded URI structure into a string, but the resultant string will be free of back references and such, it will be in its base form (or canonicalise). Wrapper functions are provided to give straight string to string operations, leaving the extrusion process transparent to the user. The code will also resolve hostnames to base form so as to avoid pages being indexed multiple times when URLs which are in effect identical, but have different host aliases or host formats are submitted.

Testing was performed by the use of a long list of URIs, both valid and invalid, which were fed through the functions to see if the results were as expected. The tests were then repeated with the code linked to ccmalloc (a memory diagnostic tool) to ensure no memory leaks were present. I would now consider the code stable, although it is not optimised, and debugging diagnostics still remain.

CHAPTER 5 – TESTING

5.0 Introduction

As detailed in the process documentation the individual modules were tested after implementation and then integrated into the final programs. These were then tested, and the results were fed back into the development cycle.

The test data used was a set of up to 32,865 pre-fetched HTML test files. The stop list was as follows :

```
a,of,about,or,all,several,also,she,among,since,an,some,and,such,are
,than,as,that,at,the,be,them,because,there,been,these,between,they,
both,this,but,those,by,to,do,toward,during,towards,each,upon,either
,v,for,v.,from,was,had,were,has,what,have,when,he,where,her,which,h
is,while,however,with,if,within,in,would,into,you,is,it,its,many,mo
re,most,must,not,on,i'll,you'll
```

The configuration was :

Abstract length	132
Title length	64
Max terms per document	20
URL chunk length	32
Word file chunk length	64
Term length	32

The testing process was integrated with collecting metrics on the project to analyse the performance. Details on implementation issues involved in this can be found in appendix D.

Timings, unless stated are CPU and kernel time used, not real time. This removes the influence of other processes running upon results, essential as I would typically run all processes on the one machine. On all timings there is a wide margin of error, timing could only be done at a granularity of 10ms, so rounding errors will have an effect.

5.1 Search Speed

The search speed will vary depending on the particular term, how many references it has, when it was added. As such the speed was estimated by searching the full test data set of 32,865 documents for each word in the UNIX dictionary, as per the stress testing described in the previous section. The server separates search and retrieve (getting information on each of the documents) operations. The average time per search was averaged out at 1ms, although has a wide margin of error because of the clock granularity. We can conclude that searches are fast, faster than 10ms in most cases, and so sufficient for this data set and ones much larger than it.

These timings are from the server side, and so I executed a typical search for the term “dog” using wget, and timed it. Wget was used rather than a browser to remove delays introduced by rendering HTML. Still, this test introduces latency from the proxy, the network, and wget itself, but gives a better indication of how quickly a user can expect a full search to execute and all details to be returned. The search and transfer of the 155 matches took 2.64s, and that was with wget running on the same machine as the server. This is perfectly acceptable, especially as the user’s screen will be filling with results as the transfer occurs.

5.2 Submission Speed

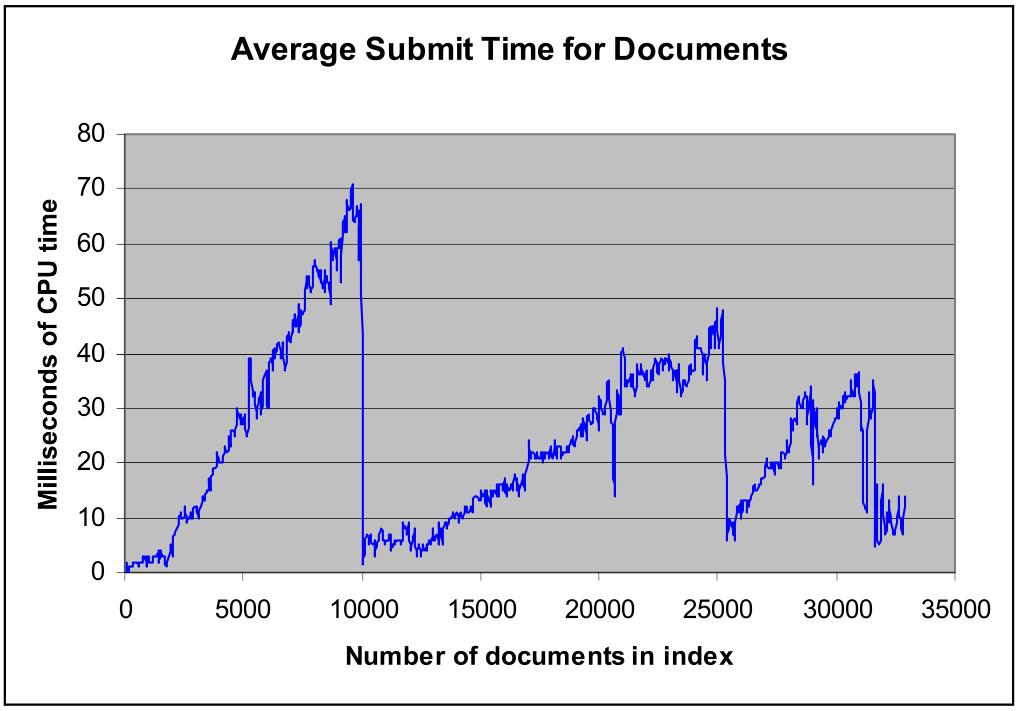


Figure 8: Graph of average submit time per document

The graph is rather an odd shape as the test data was submitted to the engine as a sequence of full websites. Some of these sites contained terms which made their way into the index for most pages, and so the chunk lists for each became long and so submissions slowed up. Once a new site was started the over-used terms were no-longer present in the index, and so submission speeds increased again.

5.3 Database Size

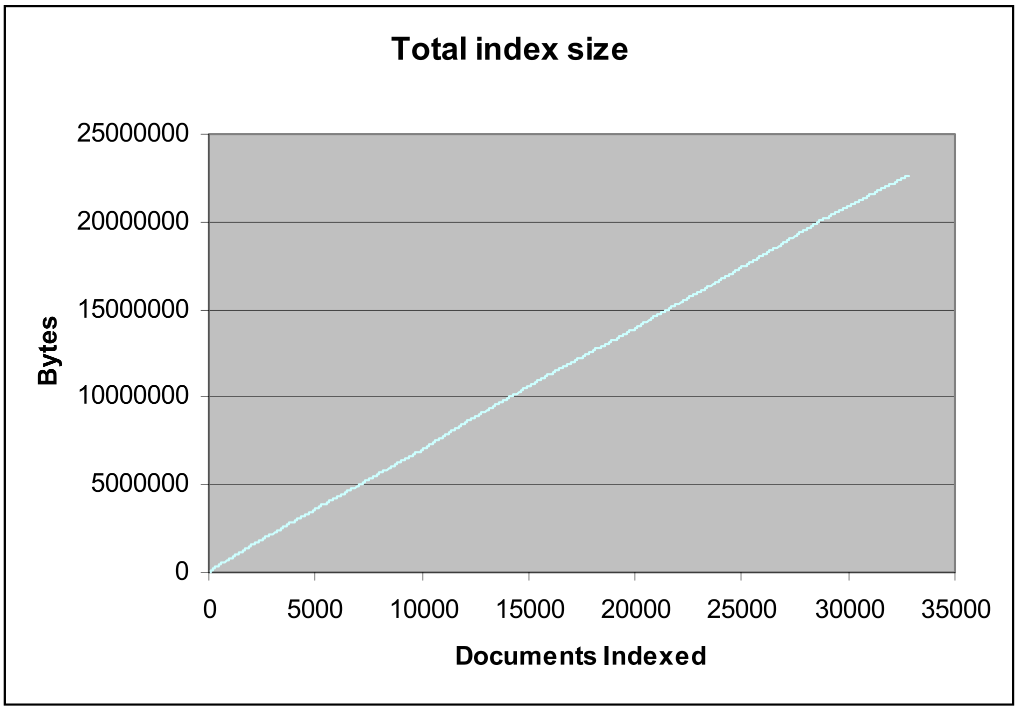


Figure 9: Graph of total index size against number of documents indexed

The total index size graph appears to be linear, but there is a subtle curvature as the number of documents increases. This is best illustrated the following graph of the index size per document as the number of documents increases.

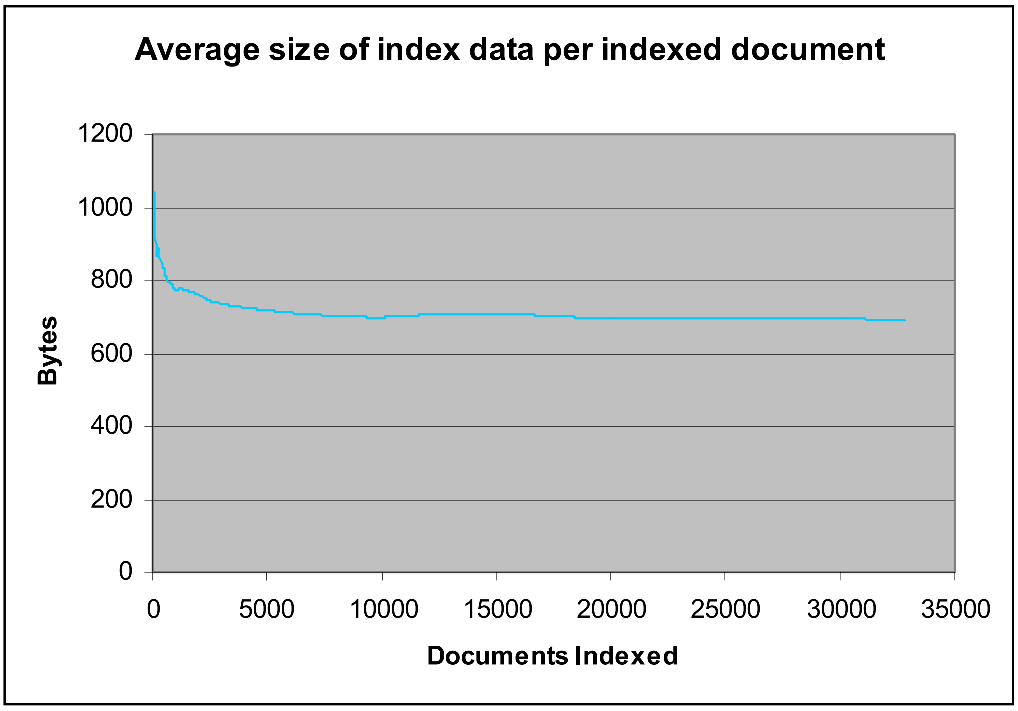


Figure 10: Graph of average size of index data per indexed document

It can be seen from the graph that the required space per document reduces quickly as common terms have chunks in the word database allocated to them and start filling up. In the test case the average settled at 695 bytes, which is well below the 1Kb figure I suggested early on in the project.

With the full test data set of HTML documents totalling 754Mb indexed the actual search engine indexes totalled a mere 21.6Mb.

5.4 Efficiency

Early on in the project I stated my intention of a tight implementation, avoiding bloat introduced by over-specified libraries and suchlike. Instead I took an approach of using “bare bones” C with only the basic UNIX extensions to implement the code. The final executables illustrate the success of this approach :

Binary file	Size (Linux 586 GCC compiled binary, approximate)
preprochtml (HTML preprocessor and document analyser)	45Kb
server (data server)	45Kb
Fetch.cgi (CGI interface)	36Kb

Table 2: Final executable sizes

Whilst in systems with hundreds of megabytes of memory the size of individual programs may seem irrelevant, but the size of the program has implications above mere memory usage. For example, a well- coded program may well be small enough to have all its core routines stored in a processor’s cache, thus considerably improving execution speed. The program size is also particularly important in the case of the server. As it forks new processes to each new connection the program is effectively duplicated, so a bloated program will consume memory very quickly. The manner in which CGI scripts are executed by web servers also implies the smaller they are the better they will run. Each time a request is made the CGI script is run, so the smaller it is the quicker it loads and the more chance it has of already being in a disc cache.

5.5 Measures of Information Retrieval Performance

It is difficult to use the measure of recall, as it depends on having a count of documents relevant to the test subject. In a small document set this would be possible as the relevant documents could be counted, but with a test set of over thirty-eight thousands documents reading them all and establishing a subject would not be practical. In a smaller document collection the engine does achieve total accuracy, assuming that the documents mention their subject search term enough for it to be indexed according to the settings.

The precision measure is more practical, as it compares the number of documents retrieved with the number of documents relevant to the query. A search for “Bill Gates” returns 265 matches, 102 of which specifically mention him, a precision of about 40% (See appendix E.1). This sounds awful, but the relevant matches were concentrated to the top ranking positions. All the documents in the results contained the terms “Bill” and/or “Gates” – I made the decision when designing the ranking algorithm not to exclude matches which did not contain all specified terms, on the grounds that it is better to return too many documents than omit those which may be relevant. The irrelevant matches tended to discuss other “Bills”, such as “Bill Hicks”, “Billing” and such. The important factor is that the engine returned relevant documents quickly and effectively.

The performance is as expected for the algorithms employed.

5.6 Profiling

As well as the usual reliability testing I needed to extract timings for evaluation of the database performance as well as for determining how to develop the code further. The first step was to attempt to profile the code using GNU gprof. There were immediate problems with this as gprof cannot time across forks, and so I would only be able to get timings from individual child processes, in other words for just one database operation.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.53     0.53                             main
  0.00      0.53     0.00      732     0.00     0.00  read_cbindex
  0.00      0.53     0.00      117     0.00     0.00  lock_file
  0.00      0.53     0.00      117     0.00     0.00  unlock_file
  0.00      0.53     0.00       43     0.00     0.00  cbtree_locate
  0.00      0.53     0.00       21     0.00     0.00  cbtree_append
  0.00      0.53     0.00       21     0.00     0.00  chunk_append
  0.00      0.53     0.00       21     0.00     0.00  chunk_retrieve
  0.00      0.53     0.00       21     0.00     0.00  expohash
  0.00      0.53     0.00       17     0.00     0.00  cbtree_getchunk
  0.00      0.53     0.00       12     0.00     0.00  allocate_next_free_chunk
  0.00      0.53     0.00       11     0.00     0.00  log_event
  0.00      0.53     0.00       10     0.00     0.00  write_cbindex
  0.00      0.53     0.00        5     0.00     0.00  cbtree_allocate
  0.00      0.53     0.00        5     0.00     0.00  chunk_allocate
  0.00      0.53     0.00        3     0.00     0.00  cbtree_open
  0.00      0.53     0.00        3     0.00     0.00  chunk_open
  0.00      0.53     0.00        1     0.00     0.00  cbtree_countentries
  0.00      0.53     0.00        1     0.00     0.00  childtask
  0.00      0.53     0.00        1     0.00     0.00  db_start
  0.00      0.53     0.00        1     0.00     0.00  db_submit_document
  0.00      0.53     0.00        1     0.00     0.00  send_tld
  0.00      0.53     0.00        1     0.00     0.00  shavtimer_child_end
  0.00      0.53     0.00        1     0.00     0.00  shavtimer_child_init
  0.00      0.53     0.00        1     0.00     0.00  shavtimer_init
  0.00      0.53     0.00        1     0.00     0.00  shavtimer_starttimer
  0.00      0.53     0.00        1     0.00     0.00  shavtimer_stoptimer
```

The profile output indicates that the process ran too quickly to get meaningful values. Normally the calls to be timed would be looped to increase the time period and therefore allow more accurate measurement. However, the profile does show the large number of calls reading the indexes produced by cbtree. This was to be expected, as it is consulted for every term addition. We can safely assume that cbtree_locate is taking most of the time because of its many calls to read_cbindex. There are several potential solutions :

1) Alter cbtree.c to become an n-ary tree, thus reducing the depth, and so increasing the search speed.
2) Alter cbtree.c to maintain the tree on disc, but actually use copy of it stored in shared memory. This would eliminate all of the slow I/O calls to disc (comparatively slow compared to shared memory whether the data has been cached or not).

The second idea would seem the most sensible to me, and would offer the greatest speed increase. There is the concern of memory usage, but after indexing 13,417 documents the indexes were found to total a mere 594K – when machines typically sport 128Mb and vast amounts of virtual memory such “waste” is hardly an issue. If it were the code could be optimised further, both the document and url files use the same index key, their tree structures are exactly the same as the database is built. The two trees could be combined, saving 12 bytes per document, reducing the index size in the example case to 436K.

5.7 Stress Testing

The system was stress tested in several ways.

Up to 32,865 test pages were fed into the indexer and therefore into the database using the following command :

```
find /export/wgot | grep .html | xargs -n 1 ./preprochtml >> out
```

The search capabilities of the engine were tested using another short shell script, which sent a query for every term in the UNIX dictionary to the web-server, stress testing the database and CGI interface. This was once again done using a UNIX shell incantation :

```
for X in `cat /usr/dict/words`; do wget -nd --directory-
prefix=/dev/null --proxy-user=cs96arh --proxy-passwd=PASSWORD
http://mipc-21/cgi-bin/fetch.cgi?q=$X; done
```

Several of these operations were run at once to increase the load and prove the concurrent operation of the server. Sample logs can be seen in Appendix E.2.

CHAPTER 6 – CONCLUSIONS

6.0 Introduction

All the figures depend on the settings the server has been configured with. For example, submission speed will increase if the size of word index chunks are increased (less seeks per search). All the settings in this evaluation are as listed in the previous chapter's introduction.

6.1 Code Functionality

The code as it stands implements the following :

- HTML parser and document analyser which reliably interprets documents, extracting words and preserving styles, as well as gathering metrics on word occurrences, abstracts, keywords and titles. Reasonably well optimised, using hash indexed binary trees to store the word occurrence data as it is gathered. The HTML parser makes use of code libraries written to translate basic HTML 2.0 entities, a simple stop-list exclusion library, and a slightly modified version of Dr. Martin Porter's reference ANSI C implementation of his stemming algorithm. It does not, at this stage, deal with such horrid things as frames, or submit URLs to any robot. Works as a network client and interfaces with the database server, successfully submitting its index entries.
- Generic chunk file library, for maintaining files containing many arbitary blocks of data of any length. The chunks are strung together into chains. Each chain is referenced by the first block. The blocked nature of this file gives random-access like speed for what would be a rather messy sequential access file (or a very large and inefficient random access one). The code can perform creation, deletion and retrieval of chunk chains, as well as providing a function to append more data to an existing chain. An extension module also allows the automatic binary tree indexing of these chunk chains by a hash key.
- Uniform Resource Locator library, deals with URLs and URIs, performing operations on both

relative and complete URLs. Will resolve them down to base form using domain name services.

- Networking interface for passing arbitary blocks of data between programs using TCP sockets. Allows the operation of multiple document analysers and CGI interfaces, and the distribution of these services across many separate computers.
- The database. Provides all needed functionality. Word lists and document attributes are stored in chunked files which are automatically indexed by binary trees. Missing functionality , as planned, is the ability to delete/update documents, and an expiry method.
- The CGI interface. Provides a useable interface for the engine, accepting queries in a HTML form and returning ranked results which are obtained from the database. The queries do not allow boolean operations as yet, due to the reasons explained in section (CGI SECTION)

As predicted the robot has not been implemented. Due to the network load involved and time considerations the engine indexes mirrors of sites kept on a local disc. This is ideal for testing and analysis purposes, as databases can be rebuilt quickly from disc.

This is enough to meet all my requirements for testing, and the code works reliably.

6.2 Known bugs list

- The HTML processor will not create an abstract text if a document has no <BODY> section. This is perfectly correct behaviour, but it leaves some documents without abstracts in search results. The processor can either return and pick up what should be the body text, or dump the document for being badly formatted.
- Two out of over thirty-two thousand HTML files tested cause the HTML pre-processor to crash. I did not have time to fix the problem, and so the pre-processor traps the error and rejects the page.
- The user interface currently reports empty queries as failures, this just needs an extra condition adding in the CGI code.

6.3 Evaluation against the ACID properties of a distributed system

As the system is distributed it seems sensible to evaluate it in relation to the ACID properties :

- Atomicity – transactions within the database are indeed all-or-nothing, and the stateless nature of the network interface ensures there are no partially completed transactions.
- Consistency – The database goes from one consistent state to another, again the atomic nature of our operations and the use of file locking prevents any inconsistent states.

- Isolation – A combination of the fact that transactions are processed on unique processes, and the use of file locking ensures isolation – no intermediate effects of transactions are visible to others.
- Durability – The server is durable in that any transaction which appears to be failing will abort, leaving the data in a recoverable state. Full recovery is not implemented as the pages can always be re-indexed at worst. It would be trivial to write a flat file recovery procedure at a later date.

6.4 Conclusion

I believe that I have achieved the project’s stated objectives and implemented a working prototype of an intelligent search engine, to prove the viability of the techniques I selected. As it stands it can feasibly index large document collections, and with the further optimisations and the implementation of the distributed system structure I described in appendix C.2 it could provide a full index of the world wide web on a small network of relatively cheap machines.

In reference to my objectives described in the first chapter :

- A large proportion of an “intelligent” search engine was developed and proved to work, thus demonstrating the techniques researched.
- Several architectures for the design, both in terms of software construction and distribution of processing were considered.
- Document analysis techniques were investigated, evaluated and implemented.
- A deeper understanding of all the techniques involved was gained.
- The implementation was efficient.

Most importantly the project proves the value of the application of information retrieval techniques to web-search technology, and the potential gains of doing so.

BIBLIOGRAPHICAL REFERENCES

Raggett, D., Le Hors, A., & and Jacobs, I. (Editors) (24th April 1998) *HTML 4.0 Specification*
W3C Recommendation

Korfhage, R. R. (1997) *Information Storage and Retrieval*
John Wiley & Sons, Inc., New York

Stevens, W. Richard (1990) *UNIX Network Programming*
Prentice Hall PTR, New Jersey

Sparck Jones, K. and Willett, P. (editors) (1997) *Readings In Information Retrieval*
Morgan Kaufmann Publishers Inc., San Franscisco, California

Levine, J. R., Mason, T. and Brown, D. (1995) *lex & yacc*
O'Reilly Associates, Inc, Sebastopol, California

Frakes, W. B. and Baeza-Yates, R. (1992) *Information Retrieval*
Prentice Hall, Upper Saddle River, New Jersey

Kernighan, B. W. and Ritchie, D. M. (1998) *The C Programming Language (2nd Edition)*
PTR Prentice Hall, Englewood Cliffs, New Jersey

Owen, F. & Jones, R. (1994) *Statistics (4th Edition)*
Pitman Publishing, London

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee (June 1999)
RFC 2616 Hypertext Transfer Protocol -- HTTP/1.1
http://www.faqs.org/rfcs/rfc2616.html

Fielding, F. (June 1995) *RFC 1808 Relative Uniform Resource Locators*
http://www.faqs.org/rfcs/rfc1808.html

Berners-Lee, T., Masinter, L. and McCahill, M. (December 1994) *RFC 1738 Uniform Resource Locators (URL)*
http://www.faqs.org/rfcs/rfc1738.html

Various/Anon (Viewed October 1999) *The Web Robots Pages*
http://info.webcrawler.com/mak/projects/robots/robots.html

Sanderson, M. (Viewed November 1999) *IDOMENEUS technology transfer server*
University of Glasgow
http://www.dcs.gla.ac.uk/idom/

Malhortra, Y. (1998) *TOOLS@WORK: Deciphering The Knowledge Management Hype*
Journal for Quality and Participation (July/August 1998) Vol. 21 No. 4, pp. 58 – 60.
Also published as *Knowledge Management for the New World Of Business*
Brint.com Institute, Fort Lauderdale
http://www.brint.com/km/whatis.htm

Peper, F. (Viewed November 1999) *What is Principal Component Analysis?*
Communication Research Laboratory of the Japanese Ministry of Post and Telecommunications
http://www-karc.crl.go.jp/avis/peper/pca.html

Porter, M. (Viewed November 1999) *The Porter Stemming Algorithm* (Official home page)
Muscat Limited
http://www.muscat.com/~martin/stem.html

Porter, M. (1980) *An Algorithm For Suffix Stripping*
In Sparck Jones, K. and Willett, P. (editors) (1997) *Readings In Information Retrieval*, pp. 313-316
Morgan Kaufmann Publishers Inc., San Francisco, California

Howe, D. (Editor) (Viewed November 1999) *Free On-Line Dictionary Of Computing (FOLDOC)*
Imperial College, London
http://foldoc.doc.ic.ac.uk/foldoc/index.html

J. Ziv and A. Lempel (May 1977) *A Universal Algorithm for Sequential Data Compression*
IEEE Transactions on Information Theory, Vol. IT-23, No. 3, May 1977, pp. 337-343

IBM Intellectual Property Network (viewed November 1999)
http://patent.womplex.ibm.com/

Welch, T. (December 1985) *US Patent 4558302: High Speed Compression and Decompression Apparatus and Method*
Sperry Corporation, New York
http://patent.womplex.ibm.com/details?&pn=US04558302__ (double underscore) (viewed November 1999)

Excalibur Corporation Homepage (viewed December 1999)
Excalibur Corporation
http://www.excalib.com/

Foster, J. (editor) (August 1994) *RFC 1689 A Status Report On Networked Information Retrieval: Tools & Groups*
University Of Newcastle-upon-Tyne
http://www.faqs.org/rfcs/rfc1689.html

Harel, D. (1992) *Algorithmics – The Spirit Of Computing (Second Edition)*
Addison-Wesley Publishing Company

Bott, F., Coleman, A., Eaton, J., Rowland, D. (1996) – *Professional Issues In Software Engineering (Second Edition)*
University Of Wales, Aberystwyth
University College London Press / Pitman Publishing

Neiderst, J. (1999) *Web Design In A Nutshell*
O'Reilly Associates, Inc, Sebastopol, California

Kirkpatrick, B (editor) (1998) *Roget’s Thesaurus of English Words & Phrases*
Penguin Books Ltd, London

Yahoo! Homepage (viewed January 2000)
Yahoo! Inc., Santa Clara, California
http://www.yahoo.com

Copernic Homepage (viewed January 2000)
Copernic Technologies Inc., Sainte-Foy, Canada
http://www.copernic.com

Lawrence, S., C. Lee Giles (April 1998) *Searching the World Wide Web*
NEC Research Institute, Princeton, NJ USA
Science Magazine, Vol. 280, 3 April 1998

Lawrence, S., C. Lee Giles (July 1999) *Accessibility of information on the web*
NEC Research Institute, Princeton, NJ USA
Nature Magazine, No. 400, 8th July 1999

Glover, E. J., Lawrence, S., Gordon, M. D., Birmingham, W. P., Lee Giles, C. (2000)
Web Search Your Way
NEC Research Institute, Princeton, NJ, USA
Artificial Intelligence Laboratory, University Of Michigan, USA
Business Administration, University Of Michigan, USA
Paper unpublished at time of writing, accepted for publication in Communications of the Association for Computing Machinery

OLCC (November 1997) *Dublin Core and Web MetaData Standards Coverage in Helsinki*
Online Computer Library Center, Inc. Dublin, Ohio, USA
http://www.oclc.org/oclc/press/971107a.htm

Kunze, J. (December 1999) *RFC 2731 Encoding Dublin Core Metadata in HTML*
http://www.ietf.org/rfc/rfc2731.txt

Inktomi Corporation Homepage (viewed January 2000)
Inktomi, San Francisco, USA
http://www.inktomi.com

Lea, G. (Viewed & Published 18th February 2000) *MS Thumps Tub For Win2k Rollout*
The Register, London, UK
http://www.theregister.co.uk/000218-000006.html

(Viewed & Published 1st March 2000) *Computer Blunders Plague Government*
BBC News Online
http://news2.thls.bbc.co.uk/hi/english/uk%5Fpolitics/newsid%5F661000/661987.stm

Yeates, D. & Cadle, J. (1996) *Project Management For Information Systems (2nd Edition)*
Financial Times/Pitman Publishing, London

Paul, R. J. (1994) *Why Users Cannot 'Get What They Want'*
International Journal of Manufacturing System Design, Vol. 1, No. 4, 1994, pp. 389-394

Sullivan, D. (Viewed 19th April 2000, Published 4th April 2000) *NPD Search and Portal Site Study*
Search Engine Watch, internet.com Corp, Darien CT, USA
http://www.searchenginewatch.com/reports/npd.html

Fenlason, J. & Stallman, R. (Viewed 29th April 2000) *GNU gprof – The GNU Profiler*
Free Software Foundation, Boston MA, USA
http://people.redhat.com/johnsonm/lad/info/gprof.html

Coulouris, G., Dollimore, J. & Kindberg, T. (1994) *Distributed Systems – Concepts & Design*
Addison-Wesley, Harlow, Essex, UK

GLOSSARY

This Glossary covers terms which may be unfamiliar to the reader. A grounding in Computer Science is assumed, so standard computer science terms are omitted as they are assumed to be common knowledge – for example it would be ridiculous to expect anyone to understand this document without knowing what the World Wide Web or Internet is, so they are not defined here.

Antonym

A word having a meaning opposite to that of another word, “Wet” is the antonym of “Dry”, for example.

Boolean Query

A query in which terms are connected by Boolean operators such as AND, OR and NOT. Extended Boolean queries may include weightings.

CGI

Common Gateway Interface, the standard method for interfacing programs with a web server.

Collision

Occurs in hashing where two different data areas give the result in the same hash code.

Compression

Use of algorithms to reduce the size of data by detecting patterns within it. Lossy compression sacrifices quality for increased compression, lossless compression loses no quality at the cost of limited compression.

Controlled Vocabulary

A restricted set of words allowed to describe documents in order to make indexing more straightforward.

Cosine Measure

Vector angle between two documents, used to measure similarity.

Crawler - See *Robot.*

Cyclic Redundancy Check (CRC)

A number (hash) derived from data, intended to allow the detection of errors within it.

Data Reduction

The process of reducing data to a smaller form whilst maintaining the necessary amount of meaning.

Directed Advertising

Advertising selected on the basis of information known about a user’s interests.

Document

A stored data record in any form.

Document Surrogate

Data used to describe a document, such as a list of keywords, an abstract etc.

Document Analysis

The process of analysing a document to determine its parts, general subject area and so on.

Document Surrogate

A limited representation of a document, in the form of keywords, an abstract and so-on.

Dublin Core

A standard for metadata elements to describe documents and information resources.

Checksum

A simple hashing method that involves summing all the data elements in a block.

Cyclic Redundancy Check (CRC)

A hashing algorithm commonly used for error detection.

Extensible Markup Language (XML)

Successor to HTML, which is extensible allowing the definition of new elements.

Effectiveness

The quality of an information system’s response to an information need.

Hashing

The process of turning a block of data into a number, either to identify that block of data or to provide error correction etc. Methods include *checksums* and *cyclic redundancy checks*.

Hashing

An incidence of two different blocks of data giving the same result when hashed.

Hash Collision

An incidence of two different blocks of data giving the same result when hashed.

Hashing Spread

The distribution of hashes of data for a given data set across the range of numbers available for a hash.

Homonyms

Words which are pronounced the same but differ in meaning, origin, or spelling.

HTML

Mark up language, text based with tags which is the main format for documents on the web.

HTTP

Hypertext Transfer Protocol, text based protocol for transferring documents. The protocol most commonly used for the transfer of HTML documents and other web material.

Information

Data – typically documents, that have been matched to a particular information need.

Information Need

The requirement to store data in anticipation of future use, or to find information in response to a current problem.

Information Retrieval (IR)

The location and presentation to the user of information relevant to a query.

Inverted Index

An index organised so each term directly identifies the documents containing that term.

Inverse Document Frequency

The logarithm of the reciprocal of the number of documents in a collection that contain that term.

Knowledge Base

Algorithms, facts, concepts and rules that form a representation of knowledge of a particular area.

Meta

A prefix taken from philosophy which means "one level higher". In web orientated terms a metasearch engine is a search engine providing another level above several search engine to provide a single interface, and metadata are elements of a document at a different level to the document text, keywords and abstracts for example.

Morphological Complexity

The level of complexity of a language's structure, word forms, inflections, derivations, compounds and such.

Natural Language Processing (NLP)

Processing of written text into a form useable by an information system.

Query

The formal expression of an information need.

PDF

Adobe’s Portable Document Format.

Recall

The proportion of relevant documents that are retrieved from a query.

Relevance Feedback

The process of using relevance ratings from users to improve the quality of future queries.

Robot

Program which downloads web pages, and follows the links within them to feed a search engine indexer.

Stop word

A word which should not be added to an index, typically because it has no meaning on its own.

Soundex

Method of translating a string into a code such that words which are pronounced the same but spelt differently (homonyms) produce a matching code

Stemming

The process of taking a word and reducing it to its morphological root, by removing tenses and suchlike. For example “stemmer”, “stemmed”, “stemming” would all be recognised as being variants of “stem”.
.

Synonym

A word or an expression that serves as a figurative or symbolic substitute for another.

Weighting

A numerical representation of a document’s relevance to a particular term or query.

White Space

A space, tab, carriage return type character that exists purely as spacing.

XML

See *Extensible Markup Language*

APPENDIX A – EVALUATION

I have personally gained a large amount of knowledge of information retrieval, databasing, web interfaces as well as improving my programming experience. Much of the code is re-useable, and so I have also gained several reliable and well-tested libraries which can be used in future large-scale indexing projects and web applications.

The research stage was interesting as I had already implemented a large database index system in C for my A-Level project, but taking a blind approach without having read up on the subject. I found that much of the research for this project was a process of formalising what I had come up with previously in terms of database construction. The matching and ranking algorithms were new to me.

I especially enjoyed implementing the project as I took a ground-up approach, and so I can legitimately claim it was "all my own work" – apart from the use of Martin Porter's reference implementation of his stemming algorithm. For this reason I had no-one else to blame if things went wrong. It was particularly satisfying when the modules and APIs integrated into a working engine – several months work of preparation of modules, and then in a short period of time a functional system is produced using the libraries I designed.

The only major hurdle within the project was actually stopping the implementation. I just resisted the temptation to spend time which was spent on writing the project up on improving the code to improve the performance of the system.

I discovered that the field of information retrieval was a vast and imprecise science, there are many equally acceptable ways of doing literally the same job.

APPENDIX B – NETWORK PROTOCOL DEFINITION

The Simple Network Interface (SNI) protocol is simple. A TCP connection is established with a server. Once this stream is established an integer type code is sent to signal the type of operation which is required. This is followed by a 32-bit word containing the length of the data block which is subsequently sent. This data block will contain all the information needed for the server to perform the operation. The client then waits for a response code, followed by another length word, and a returned data block. It then drops the connection and processes the returned data. So :

```
Client:    Establish connection
Client:    [4 bytes type code]
Client:    [4 bytes length]
Client:    [<length> bytes of data]
Server:    [4 byte response code]
Server:    [4 byte response length]
Server:    [<response length> data]
Client:    Disconnect session
```

Operation codes and response codes have been given unique numbers for the sake of clarity, but they are in fact two distinct sets.

The operation codes currently defined for this project are :

SNI_MSGTEST (1)

Debugging operation, prints the attached string to the server log.

Data type: String

Returns: SNI_OK

SNI_DOCSUBMISSION (100)

Submits a document to the server.

Data type: A PreProcDoc structure followed by PreProcDoc.indexedwords WordEntry elements. At the end of this (offset = sizeof(PreProcDoc)+(ppd.indexedwords*sizeof(WordEntry))) there is a string, which contains the url. Definitions for WordEntry and PreProcDoc follow.

```
typedef struct
{
  char word[32];
  int occurrences;
  int weighting;
} WordEntry;

typedef struct
{
  char title[80];     /* document title */
  char abstract[132]; /* document abstract */
  time_t time;        /* time of indexing */
  int doclength;      /* original document length */
  int wordcount;      /* count of words indexed */
  int indexedwords;   /* count of weight/word pairs to add to index */
  /* WordEntry wlist;  * there follows a list of [indexedwords]
                       * WordEntry elements */
  /* char *url[]       * and then the source url */
} PreProcDoc;
```

Returns: SNI_OK or SNI_FAIL.

SNI_GETDOCRECORD (101)

Requests a DocumentRecord structure.

Data type : a simple 32 bit integer containing the hash of the base form URL of the document – in other words the document's identifier.

Returns: SNI_FAIL or SNI_DOCRECORD.

SNI_GETDOCCHUNK (102)

Requests a DocumentRecord structure. NB: This function was largely for debugging, it is best to access using SNI_GETDOCRECORD.

Data type : a simple 32 bit integer containing the chunk_id of the DocumentRecord.

Returns: SNI_FAIL or SNI_DOCRECORD.

SNI_QUERY (103)

Starts a multi-term query. Returns a SNI_QUERYRESULTS response if successful.

Data type : a 32 bit integer containing the number of terms. This is immediately followed by that number of fixed length query terms as strings of TERMSTRINGLENGTH.

Returns: SNI_FAIL or SNI_QUERYRESULTS.

SNI_MSGINFO (104)

Request server stats, returned as SNI_OK message or SNI_FAIL. SNI_OK will have server stats attaeched a pre-formatted HTML to insert into document.

The response codes currently defined are :

SNI_OK (200)

Generic OK, can optionally be followed by a string.

Data type: (optional) String.

SNI_FAIL (201)

Generic failure, any data is an error message string.

Data type: (optional) String.

SNI_DOCRECORD (202)

Document data return in response to a SNI_GETDOCRECORD or SNI_GETDOCCHUNK message.

Data type: A DocRecord structure, followed by the URL string.

```
typedef struct
{
  int  urlhash;      /* hash of lowest form url */
  int  sourceurl;    /* chunk pointer to url in urls file - stored like this
                      * as urls vary drastically in length */
  char title[80];    /* document title */
  char abstract[132];/* first 131 bytes of text, abstract, whatever */
  time_t time;       /* time document was indexed */
  int  length;       /* length of original document in bytes */
} DocumentRecord;
```

SNI_QUERYRESULTS (203)

Returns the document identifiers and weightings in response to a SNI_QUERY.

Data type: a 32 bit integer giving the number of matches, followed by (for each match) a 32 bit integer document identifer (for use with SNI_GETDOCCHUNK) and a 32 bit integer weighting value. The list is pre-sorted "best-first".

The macro aliases for the transfer types are defined in "sni.h".

APPENDIX C – FUTURE WORK

C.1 General

Features not implemented as yet :

- Full locking and concurrency for database entries rather than simple file locking. This would improve performance under load considerably.
- Page expiry. Use of page expiry headers in HTTP responses and/or page meta tags to schedule re-fetching and indexing of a page.
- Support for frames, not only indexing them, but when each frame part is indexed its link should be to the parent page which defined its position. Potentially this parent page could be a child of another page, and so the process could become quite complex.
- Pagination of results. Currently the user is instantly sent all matches for a query, without splitting the results into shorter “pages”. This could require a little more CGI programming. Potentially the CGI code could assign a query ID to the session, this could be used as an ID for a temporary cache of the results. The fact that the database only returns ID numbers for matches and leaves the client to request data for each page increases the potential efficiency of this.
- Use of PICs ratings to provide levels of suitability to allow the filtering of “unsuitable” content from search results depending on user preferences.
- Automatic fetching of pages.
- Recognition of language attributes, perhaps to facilitate a multi-lingual database, with different stemming algorithms for each language.
- Support for complex character sets, using UniCode and the like.

The HTML pre-processor requires further testing, as there are some disturbing impressions of HTML on the web, which include such foulness as random control characters, mismatched tags and such. The pre-processor could reject such pages, as if the coding is that poor the page content is unlikely to be much use either.

It would be interesting to do future work with large document collections in order to prove the scalability of the system as described in this text. Whilst doing this the server could be enhanced so it gathers its own metrics on the document collection, covering such things as the average term length, average URL length and such. This way the configurable parameters of the program, such as chunk sizes, could be altered to give optimal performance. This work should also include obtaining reliable timing and performance metrics to allow the comparison of algorithms, methods and configurations.

The use of non-weighted queries made the implementation of the ranking procedure rather trivial. The addition of a weighted query parser, and the inclusion of weights in the transfer structures etc proved too much work for the final stages of the project. This was unfortunate, but unavoidable. As such I have detailed in the query code (db.c function db_lookup) how such methods would be implemented.

A final idea is that the search engine could weight towards query terms which are nouns rather than adjectives when searching. This way a search for "small dogs" would prefer pages talking specifically about dogs, rather than pages more concerned with small things in general.

C.2 Scalability Considerations

One of the stated aims of this work was to produce a system which didn't require the terabytes of storage and huge processing resources of systems such as AltaVista. Still, there is the potential of a database becoming too large to be handled by one machine efficiently.

chunk_id is currently defined as being an int, which under gcc is equivalent to a 32 bit signed integer. This will allow 4.3 billion chunks (where 1 billion is 10^9 – the traditional British definition is 1 billion is 10^{12}, but conventions seems to be falling on 10^9). The use of chunks in the prototype means than there is an effective limit of 4.3 billion documents, and 4.3 billion index terms. This would seem adequate, considering the estimation of there being 800 million publicly accessible web pages in 1999 (Lawrence, Lee Giles 1999). If the limit did prove restrictive chunk_id could be redefined as a long long, which under recent versions of gcc is equivalent to a 64 bit integer. This would allow 1.84×10^{19} (1.84×10^9 billion) documents and 1.84×10^{19} terms to be indexed. This would certainly be future-proof, but there would be serious practical problems in maintaining a database of such a colossal size.

The chunk code could also be modified to split its data and trees according to their hashes to reduce the overall size of each. This technique would not make much difference to response times, in itself, but when each separate tree is stored on a dedicated physical disc the physical access time is vastly reduced. The binary trees could also potentially be kept in memory rather than searched on disc, whereas in the prototype we rely on the cache. This is highly inefficient in terms of memory usage, but would be quite desirable if money were no object.

As the database increases in size the need for distribution becomes more necessary, this is partially implemented in the prototype, in that the document analysers and fetchers may run on machines separate from the database server. The use of fork() in the server also makes it scalable in symmetric multi-processor systems. However, any final implementation would also have the database split. The mechanism I would propose for this would be to spread term and weighting data over several machines according to the term's hash. Consider we have *n* servers. The submission process would select the server to submit each term to by the following rule :

server = termhash MOD servers

where MOD is the modulus operation. This way the spread of terms across servers will be roughly even. Equally the query interface would query each server as appropriate for each term and collate the results.

The actual document data, would be stored in a single database stored on a dedicated server – although if load on this machine proved to be a problem the data could be mirrored on several machines.

I considered several models for providing an interface to the many servers – one option would be to have a machine running a program to distribute the queries, collate them and return results in a manner similar to the prototype. I decided this was not appropriate as it would again impose a reliance on a single server – whilst the data may be distributed across many machines one machine still has to deal with all the incoming queries. Instead I decided this sort of functionality should be implemented in the programs communicating with the servers, through a library which would provide all of this functionality transparently.

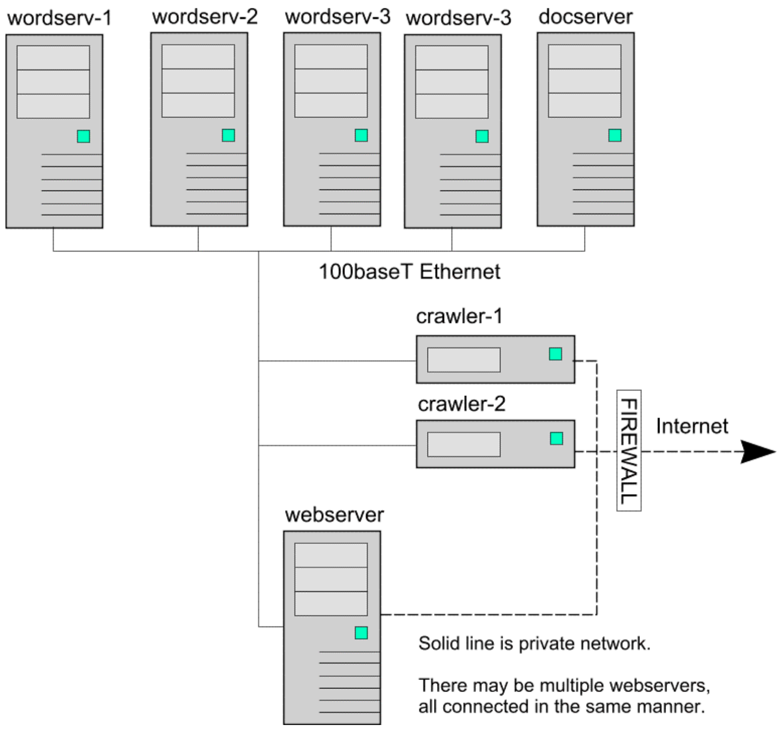


Figure 11: Sample network topology for distributed search engine

The servers for the search engine should be on a dedicated 100baseT network, isolated from all other network activity and only linked to the web servers running the CGI interface and the crawler machines.

In terms of specifications the document data server would have to be the most powerful machine, using a RAID array for storage. The disc requirements for servers would not actually be that great, even if the average index data per document were 5k. That is only 4Gb for the entire web as it stood in 1999. This server will be under the most load assuming it is not mirrored. The word/weighting servers can be of a lesser specification as they have their load spread across several machines. The crawler machines can be bog-standard PCs, they have little memory, storage or processor requirements, as their major limitation is how fast they can fetch pages over the web. Each crawler would run many crawler processes to maximise throughput.

I am confident that in combination with the techniques described in this chapter any eventual system, given appropriate investment in hardware and software, will scale to any application. The only hard and fast limit would be the 100Mb/s limit of the internal network, but again, this could be split into multiple sections and more servers added. However, I doubt such measures would ever be required as I would expect an engine equivalent to say Google, to run quite effectively on a couple of reasonably specified servers with RAID, and a standard PC to crawl the web

APPENDIX D – IMPLEMENTATION OF TIMERS FOR PROJECT METRICS

To time operations performed by the server I implemented a set of timing functions that used the UNIX clock to time each operation and return an average operation duration on request. This is straightforward, but not when each operation is forked. The fork takes a copy of the structures and uses them, and so as a result any timer implemented normally would end up either not timing anything at all or at best timing one operation.

To solve the problem I had to use shared memory to store the timing variables for each attribute. This introduced a fair level of complexity as the memory had to be managed effectively. Each child had to register its interest in the data when it started, and also absolutely positively had to be release it when it was finished with, as shared memory if it isn't released by all its users will hang around until the next reboot. I ensured that memory was freed by adding an atexit() handler.

This was fine, and I had average timing statistics. However I was timing against a real-time clock. There are two significant problems with this. The first is that the granularity of the clock is such that it can't accurately measure the time taken for most of the operations I intended to time. Secondly the real world time was not particularly useful anyway, as results will vary depending on the general system load. I needed a timer which would give me a measurement of CPU time used by the actual code.

The standard UNIX "ps" program lists the CPU usage of a process, so I consulted its source code after scouring the documentation to look for an API. Trawling through the /proc virtual filesystem I found the values I needed, only to discover that both system and kernel processor usage per process was only timed to a resolution of one second. This was no use, as our processes typically complete in less than half a second.

The only other solution was to use itimers, which can be set to call at a regular interval. They can be used to implement virtual time. They are timers which send an alarm signal to at a fine-grain interval (currently 10ms), and can be clocked against real, virtual or profile time. Virtual time is counted only when the process is executing, and profile time is counted when the process is executing, or the kernel is executing operations on behalf of the process. Obviously the process of having these alarms will cause a certain amount of load, and so the timers would be omitted from production code.
Each child process sets its own itimer and keeps its own virtual timer by incrementing an integer on every signal.

Virtual timers worked fine, but many operations were still too slow to time, and the rounding errors introduced by the clock granularity meant the average time for quick operations such as document information retrieval is logged as being 0ms. Whilst the timing obviously isn’t accurate, we can certainly assume it to be far less than 10ms, and in our case that can certainly be assumed to be “fast enough”.

So, the eventual implementation was :

- Parent (initial server process) calls shavtimer_init() which sets up the shared memory block, and registers an atexit() handler to de-allocate the block. It also keeps a record of the parent’s process ID. :

```
int shavtimer_init()
{
  /* keep a copy of parent process id so we know what to do on atexit */
  ppid=getpid();

  /* create the shared memory segment */
  if((shmid = shmget(SHAVSHM_KEY,
                     sizeof(SharedAverageTimer)*(int)shavt_count,
                     PERMS | IPC_CREAT)) < 0 )
  {
    log_event("shavtimer_init: Failed to create shared memory segment, attempting to open
and clear any existing shared memory block");

    /* create the shared memory segment */
    if((shmid = shmget(SHAVSHM_KEY,
                       sizeof(SharedAverageTimer)*(int)shavt_count,
                       0)) < 0 )
    {
      log_event("shavtimer_init: Failed to find any previously allocated shared memory");
      return -1;
    }
  }

  /* register our exit handler */
  atexit(shavtimer_exithandler);

  /* attach it */
  if((shavt = (SharedAverageTimer *)shmat(shmid, (char *)0, 0))
     == (SharedAverageTimer *)-1)
  {
    log_event("shavtimer_init: Failed to attach shared memory");
    return -1;
  }

  memset(shavt,'\0',sizeof(SharedAverageTimer)*(int)shavt_count);

  return 0;
}
```

- Child process is forked for a connection, and callls shavtimer_child_init() which registers interest in shared memory, sets up the itimer, and registers an alarm handler to deal with the SIGPROF alarms. As the child process has been forked it shares the atexit() handler of the parent.

```
int shavtimer_child_init()
{
#ifdef VIRTUALTIME
  struct itimerval itv,oval;
#endif

  /* create the shared memory segment */
  if((shmid = shmget(SHAVSHM_KEY,
                     sizeof(SharedAverageTimer)*(int)shavt_count,
                     0)) < 0 )
  {
    fprintf(stderr,"shavtimer_child_init: Failed to claim shared memory\n");
    return -1;
  }
  /* attach it */
  if((shavt = (SharedAverageTimer *)shmat(shmid, (char *)0, 0))
              == (SharedAverageTimer *)-1)
  {
    fprintf(stderr,"shavtimer_child_init: Failed to attach shared memory\n");
    return -1;
  }

#ifdef VIRTUALTIME
  /* we need to set up an itimer on our process, so we can count
   * virtual time - register our signal handler and set timer
   */
  millisecondclock=0;
  signal(SIGPROF,(void (*)())sig_profalarm);

  itv.it_interval.tv_sec=0;
  itv.it_interval.tv_usec=10000; /* 10ms */
  itv.it_value.tv_sec=0;
  itv.it_value.tv_usec=10000; /* 10ms */
  setitimer(ITIMER_PROF,&itv,&oval);
#endif

  return 0;
}
```

- Alarm handler is called periodically to keep time, which also sets an alarm for the next 10ms :

```
/* SIGPROF signal handler */
void sig_profalarm()
{
  struct itimerval itv,oval;

  millisecondclock+=10;
  itv.it_interval.tv_sec=0;
  itv.it_interval.tv_usec=1000;
  itv.it_value.tv_sec=0;
  itv.it_value.tv_usec=1000;
  setitimer(ITIMER_PROF,&itv,&oval);
}
```

- When either the parent or child ends the atexit function is called, which compares the process ID with the one we retained as being that of the parent. It either calls the shavtimer_end() routine for a parent process :

```
/* close shared timers -- called in parent process, returns
 * -1 on failure
 */
int shavtimer_end()
{
  if(shmid)
  {
    /* detach it */
    if(shmdt(shavt) < 0)
    {
      fprintf(stderr,"shavtimer_end: Failed to detach shared memory\n");
      return -1;
    }

    /* remove it */
    if(shmctl(shmid,IPC_RMID, (struct shmid_ds *) 0) < 0)
    {
      fprintf(stderr,"shavtimer_end: Failed to remove shared memory\n");
      return -1;
    }
  }

  shmid=0;

  return 0;
}
```

Or the shavtimer_child_end process for a child process :

```
/* close shared timers -- called in child process, returns
 * -1 on failure
 */
int shavtimer_child_end()
{
  if(shmid)
  {
    /* detach it */
    if(shmdt(shavt) <0)
```

```
      {
        fprintf(stderr,"shavtimer_child_end: Failed to detach shared memory\n");
        return -1;
      }
    }

    shmid=0;

    return 0;
  }
```

The full code can be seen in the source listings. The code is still #define-able to give results in real time rather than virtual/profile time.

APPENDIX E – SAMPLE TEST OUTPUTS

Many test runs were performed which produced far too much data to reproduce here. The printed tests are selected from the overall test outputs to prove important attributes of the system.

E.1 Output from query for “Bill Gates”

```
Search for terms : bill, gates (bill, gate)
265 matches to query.
THE REGISTER: Analysis: How Bill Gates discovered the Web (Weight 864)
Posted 06/12/98 4:03pm by John Lettice Analysis: How Bill Gates discovered
the Web What did Microsoft think about the Web, and when...
http://www.theregister.co.uk/981206-000001.html - 24633 bytes, fetched on
Wed May 3 13:03:49 2000
THE REGISTER: Gates video 'not a beautiful thing to watch,' say (Weight
296)
Posted 17/11/98 5:14pm by Graham Lea Gates video 'not a beautiful thing to
watch,' says Microsoft brief Even Microsoft legal adviso...
http://www.theregister.co.uk/981117-000025.html - 40552 bytes, fetched on
Wed May 3 13:04:37 2000
THE REGISTER: Gates TV: Killer apps and hit teams remembered, s (Weight
254)
Posted 16/12/98 10:22am by Graham Lea Gates TV: Killer apps and hit teams
remembered, sort of In yesterday's video Bill Gates was c...
http://www.theregister.co.uk/981216-000007.html - 38065 bytes, fetched on
Wed May 3 13:04:48 2000
THE REGISTER: DoJ springs trap for Bill Gates (Weight 216)
Posted 20/10/98 9:39am by Graham Lea and John Lettice DoJ springs trap for
Bill Gates The US Department of Justice aimed straight f...
http://www.theregister.co.uk/981020-000002.html - 12486 bytes, fetched on
Wed May 3 13:03:55 2000
THE REGISTER: Gates moans to press in PR spectacle (Not) (Weight 208)
Posted 08/12/98 9:42am by John Lettice Gates moans to press in PR spectacle
(Not) The government is "trying to turn this into a PR ...
http://www.theregister.co.uk/981208-000004.html - 12842 bytes, fetched on
Wed May 3 13:04:50 2000
THE REGISTER: Gates: He's just a guy who can't say N... N... (Weight 180)
Posted 16/11/98 9:42pm by John Lettice Gates: He's just a guy who can't say
N... N... The Bill Gates video excerpt has become a wel...
http://www.theregister.co.uk/981116-000021.html - 12295 bytes, fetched on
Wed May 3 13:04:39 2000
THE REGISTER: Gates steps down as MS CEO -- DoJ deal pending? (Weight
128)
MONDAY MAY 1ST 2000 The Register Bulletin Board Microsoft on Trial
Useful Links The Quick Guide to Register Jargon Flame of the Wee...
http://www.theregister.co.uk/000114-000003.html - 15250 bytes, fetched on
Wed May 3 13:03:31 2000
THE REGISTER: Gates under the grill - the full transcript (Weight 110)
Posted 10/11/98 5:04pm by Graham Lea Gates under the grill - the full
transcript Below we publish the first full transcript of Bill...
http://www.theregister.co.uk/981110-000022.html - 20210 bytes, fetched on
Wed May 3 13:04:42 2000
THE REGISTER: Gates' memory problems causing concern (Weight 108)
Posted 20/11/98 4:37pm by Graham Lea Gates' memory problems causing
concern The Gates deposition is a cause for concern in the Micr...
http://www.theregister.co.uk/981120-000025.html - 21078 bytes, fetched on
Wed May 3 13:04:36 2000
THE REGISTER: Urinate on Java? Nope, Bill doesn't know what thi (Weight
84)
Posted 03/12/98 9:12am by Graham Lea Urinate on Java? Nope, Bill doesn't
know what this means... The bailiff in the Washington cou ...
http://www.theregister.co.uk/981203-000003.html - 28262 bytes, fetched on
Wed May 3 13:04:52 2000
THE REGISTER: Gates paves way for opening up Windows source (Weight
80)
MONDAY MAY 1ST 2000 CeBIT 2000: full coverage The Register Bulletin
Board Microsoft on Trial Useful Links The Quick Guide to Regist...
http://www.theregister.co.uk/000218-000002.html - 14322 bytes, fetched on
Wed May 3 13:03:23 2000
THE REGISTER: The day Bill Gates screamed IBM's house down (Weight
80)
Posted 08/06/99 12:29pm by John Lettice The day Bill Gates screamed IBM's
house down Bill Gates went so ballistic in a phone call t...
http://www.theregister.co.uk/990608-000020.html - 11923 bytes, fetched on
Wed May 3 13:04:02 2000
THE REGISTER: Bill speaks on integration, lawyering and Linux (Weight 70)
MONDAY MAY 1ST 2000 Full Register Comdex Fall Coverage The Register
Bulletin Board Microsoft on Trial Useful Links The Quick Guide ...
http://www.theregister.co.uk/991111-000029.html - 16662 bytes, fetched on
Wed May 3 13:03:36 2000
THE REGISTER: Gates tapes: latest transcript (Weight 70)
Posted 07/01/99 12:01pm by Graham Lea Gates tapes: latest transcript In the
interest of completeness, we can now give the full text...
http://www.theregister.co.uk/990107-000008.html - 17460 bytes, fetched on
Wed May 3 13:05:02 2000
THE REGISTER: Bill Gates devil numerologist can't count (Weight 64)
MONDAY MAY 1ST 2000 The Register Bulletin Board Microsoft on Trial
PR alert -- we publish our tariff Intel's Official Guide to the ...
http://www.theregister.co.uk/991101-000034.html - 12298 bytes, fetched on
Wed May 3 13:05:31 2000
Angst (Weight 63)
Select from listnme.com TicketshopNewsReviewsSingles ArchivePick Of The
MonthGigsChartsTop 50 of 19991998's Top 50 Albums1998's Top...
http://www.nme.com/board/angst/home.html - 295326 bytes, fetched on Wed
May 3 14:14:10 2000
THE REGISTER: 'I read all my email' - shock Gates admission (Weight 60)
Posted 16/03/99 9:39am by John Lettice 'I read all my email' - shock Gates
admission Months after his toe-curling, stonewalling vid...
http://www.theregister.co.uk/990316-000006.html - 10962 bytes, fetched on
Wed May 3 13:04:09 2000
THE REGISTER: Analysis: Selling the Web to Bill (Weight 60)
Posted 13/12/98 9:51pm by John Lettice Analysis: Selling the Web to Bill Ben
Slivka is one of the finds of the Microsoft antritrust...
http://www.theregister.co.uk/981213-000006.html - 18126 bytes, fetched on
Wed May 3 13:04:49 2000
THE REGISTER: Gates video - Microsoft's 'hit team' to get IBM i (Weight 56)
Posted 15/12/98 7:13pm by John Lettice Gates video - Microsoft's 'hit team' to
get IBM into line Back in 1994 Bill Gates asked his ...
http://www.theregister.co.uk/981215-000025.html - 12098 bytes, fetched on
Wed May 3 13:04:48 2000
THE REGISTER: Chase denies Gates said 'How much to screw Netsca
(Weight 50)
Posted 12/02/99 8:12am by John Lettice Chase denies Gates said 'How much
to screw Netscape' Bill Gates never said to AOL executives...
http://www.theregister.co.uk/990212-000002.html - 11726 bytes, fetched on
Wed May 3 13:04:18 2000
THE REGISTER: Torch the emails? Intel exec claims Gates conside (Weight
50)
Posted 11/11/98 11:46am by John Lettice Torch the emails? Intel exec claims
Gates considered it Showing less than his usual prescie...
http://www.theregister.co.uk/981111-000008.html - 13600 bytes, fetched on
Wed May 3 13:04:41 2000
Angst (Weight 50)
Select from listnme.com TicketshopNewsReviewsSingles ArchivePick Of The
MonthGigsChartsTop 50 of 19991998's Top 50 Albums1998's Top...
http://www.nme.com/board/angst/index.html - 302931 bytes, fetched on Wed
May 3 14:14:09 2000
THE REGISTER: Microsoft on Trial: January 99 (Weight 49)
Posted 03/09/99 1:58pm by Team Register Microsoft on Trial: January 99
Back to the main Microsoft on Trial page Stories from... Dec...
http://www.theregister.co.uk/990903-000019.html - 20286 bytes, fetched on
Wed May 3 13:04:24 2000
THE REGISTER: Gates article contradicts MS lawyers - again (Weight 48)
Posted 03/06/99 4:38am by John Lettice Gates article contradicts MS lawyers -
again As heartlessly and repeatedly predicted here, B...
http://www.theregister.co.uk/990603-000006.html - 11444 bytes, fetched on
Wed May 3 13:04:04 2000
THE REGISTER: Microsoft on Trial: December 1998 (Weight 48)
Posted 03/09/99 1:52pm by Team Register Microsoft on Trial: December 1998
Back to the main Microsoft on Trial page Stories from... ...
http://www.theregister.co.uk/990903-000017.html - 16203 bytes, fetched on
Wed May 3 13:04:25 2000
THE REGISTER: Intel writes lousy software, says Gates (Weight 48)
Posted 09/11/98 9:47pm by John Lettice Intel writes lousy software, says
Gates Here's a puzzle. One of the long term Wintel buddies...
http://www.theregister.co.uk/981109-000024.html - 12083 bytes, fetched on
Wed May 3 13:04:42 2000
THE REGISTER: Government victimising Gates, says Microsoft (Weight 48)
Posted 04/11/98 9:06am by Graham Lea Government victimising Gates, says
```

Microsoft Bob Herbold, Microsoft COO, said in his introduct...
http://www.theregister.co.uk/981104-000001.html - 11233 bytes, fetched on Wed May 3 13:04:44 2000
THE REGISTER: What the hell is... the UK's RIP Bill (Weight 48)
MONDAY MAY 1ST 2000 Cash Register BOFH 2000: Kit and Caboodle The Register Bulletin Board Microsoft on Trial Links to Web-related S...
http://www.theregister.co.uk/000314-000016.html - 20841 bytes, fetched on Wed May 3 13:06:46 2000
THE REGISTER: Gates squirms on screen as memory problems re-eme (Weight 46)
Posted 03/11/98 9:29am by Graham Lea Gates squirms on screen as memory problems re-emerge Apple VP Avie Tevanian's cross-examinatio...
http://www.theregister.co.uk/981103-000004.html - 15854 bytes, fetched on Wed May 3 13:04:44 2000
THE REGISTER: Gates TV -- the final instalment (Weight 46)
Posted 18/01/99 4:42pm by Graham Lea Gates TV -- the final instalment Among the thousands of pages of exhibits and depositions rele...
http://www.theregister.co.uk/990118-000012.html - 14889 bytes, fetched on Wed May 3 13:04:57 2000
THE REGISTER: 'Leaked' Gates email spins party line on AOL (Weight 40)
Posted 02/12/98 9:22am by John Lettice 'Leaked' Gates email spins party line on AOL One night last month Bill Gates phoned an Assoc...
http://www.theregister.co.uk/981202-000003.html - 12752 bytes, fetched on Wed May 3 13:03:55 2000
THE REGISTER: Gates trails antitrust deal - but is he stalling (Weight 40)
Posted 25/03/99 11:00am by Graham Lea Gates trails antitrust deal - but is he stalling again? It's quiz time again: what have Sir D...
http://www.theregister.co.uk/990325-000005.html - 15197 bytes, fetched on Wed May 3 13:04:08 2000
Buffalo Bills (Weight 38)
ADVERTISEMENT Go to: Fun & Games - Translator - Word of the Day - Help Search: the Web Dictionary Thesaurus for Web Di...
http://www.dictionary.com/Dir/Sports/Football/Professional/Teams/Buffalo_Bills/index.html - 7766 bytes, fetched on Wed May 3 13:25:54 2000
Billing (Weight 38)
ADVERTISEMENT Go to: Fun & Games - Translator - Word of the Day - Help Search: the Web Dictionary Thesaurus for Web Di...
http://www.dictionary.com/Dir/Computers/Software/Accounting/Billing/index.html - 8443 bytes, fetched on Wed May 3 13:34:24 2000
THE REGISTER: Gates fingered as running predatory strategy (Weight 36)
Posted 06/01/99 2:33pm by John Lettice Gates fingered as running predatory strategy The DoJ's final witness, MIT economist Franklin...
http://www.theregister.co.uk/990106-000007.html - 11783 bytes, fetched on Wed May 3 13:05:03 2000
THE REGISTER: Bill Gates is the devil - - whacky numerologist ' (Weight 36)
Posted 27/10/99 5:56pm by Thomas Greene Bill Gates is the devil - - whacky numerologist 'proves' If you take all the letters in Bi ...
http://www.theregister.co.uk/991027-000024.html - 8963 bytes, fetched on Wed May 3 13:05:32 2000
THE REGISTER: Gates moans to press about DoJ deposition tactics (Weight 36)
Posted 22/11/98 5:54pm by John Lettice Gates moans to press about DoJ deposition tactics Last week as Bill Gates' video testimony w...
http://www.theregister.co.uk/981122-000001.html - 14091 bytes, fetched on Wed May 3 13:03:55 2000
THE REGISTER: Big Brother Bill faces Select Committee storm (Weight 30)
MONDAY MAY 1ST 2000 Cash Register BOFH 2000: Kit and Caboodle The Register Bulletin Board Microsoft on Trial Links to Web-related S...
http://www.theregister.co.uk/000313-000001.html - 17036 bytes, fetched on Wed May 3 13:06:46 2000
THE REGISTER: Will Gates take the stand? (Weight 30)
Posted 23/11/98 6:38am by Graham Lea Will Gates take the stand? Gates' interview with Associated Press (Earlier Story), and his com...
http://www.theregister.co.uk/981123-000004.html - 11279 bytes, fetched on Wed May 3 13:04:36 2000
THE REGISTER: 'Gates unplugged' session sheds light on 1995 str (Weight 30)
Posted 11/11/98 1:09pm by Graham Lea 'Gates unplugged' session sheds light on 1995 strategy Bill Gates gave a wide-ranging briefing...
http://www.theregister.co.uk/981111-000015.html - 13190 bytes, fetched on Wed May 3 13:04:41 2000
THE REGISTER: Gates to Intel: stop competing with Microsoft (Weight 30)
Posted 10/11/98 6:19pm by Graham Lea Gates to Intel: stop competing with Microsoft Microsoft had been a little reluctant to disclos...
http://www.theregister.co.uk/981110-000023.html - 14108 bytes, fetched on Wed May 3 13:04:42 2000
THE REGISTER: Gates hits Washington in serial lobbying schmooze (Weight 28)
MONDAY MAY 1ST 2000 BOFH 2000: Kit and Caboodle The Register Bulletin Board Microsoft on Trial Useful Links The Quick Guide to Regi...
http://www.theregister.co.uk/000406-000013.html - 15103 bytes, fetched on Wed May 3 13:03:08 2000
THE REGISTER: Microsoft draws blood in Netscape battle (Weight 25)
Posted 21/10/98 9:53pm by John Lettice Microsoft draws blood in Netscape battle Microsoft finally drew blood this morning when it p...
http://www.theregister.co.uk/981021-000023.html - 12533 bytes, fetched on Wed May 3 13:04:30 2000
THE REGISTER: Microsoft lawyer claims AOL deal makes lawsuit de (Weight 25)
Posted 23/11/98 7:08pm by John Lettice Microsoft lawyer claims AOL deal makes lawsuit dead parrot Microsoft general counsel William...
http://www.theregister.co.uk/981123-000020.html - 12403 bytes, fetched on Wed May 3 13:04:35 2000
THE REGISTER: Microsoft on Trial (Weight 25)
MONDAY MAY 1ST 2000 The Register Bulletin Board Useful Links The Quick Guide to Register Jargon Flame of the Week The Register's My...
http://www.theregister.co.uk/981020-000020.html - 55284 bytes, fetched on Wed May 3 13:03:07 2000
THE REGISTER: Opposition mounts against UK's Big Brother Bill (Weight 22)
MONDAY MAY 1ST 2000 Cash Register BOFH 2000: Kit and Caboodle The Register Bulletin Board Microsoft on Trial Links to Web-related S...
http://www.theregister.co.uk/000306-000020.html - 14003 bytes, fetched on Wed May 3 13:06:46 2000
THE REGISTER: Microsoft on Trial: November 98 (Weight 22)
Posted 03/09/99 1:44pm by Team Register Microsoft on Trial: November 98 Back to the main Microsoft on Trial page Stories from... Ja...
http://www.theregister.co.uk/990903-000015.html - 21164 bytes, fetched on Wed May 3 13:04:25 2000
THE REGISTER: Judge damns Gates as 'unresponsive witness' (Weight 22)
Posted 20/11/98 10:28am by John Lettice Judge damns Gates as 'unresponsive witness' Microsoft chairman Bill Gates was yesterday cha...
http://www.theregister.co.uk/981120-000002.html - 11198 bytes, fetched on Wed May 3 13:04:37 2000
THE REGISTER: Gates memos show how Microsoft puts screws on Int (Weight 22)
Posted 10/11/98 12:05pm by John Lettice Gates memos show how Microsoft puts screws on Intel Threats? What threats? Bill Gates' clai...
http://www.theregister.co.uk/981110-000004.html - 13800 bytes, fetched on Wed May 3 13:04:43 2000
Finance (Weight 20)
ADVERTISEMENT Go to: Fun & Games - Translator - Word of the Day - Help Search: the Web Dictionary Thesaurus for Web Di...
http://www.dictionary.com/Dir/Computers/Internet/WWW/Web_Portals/Netscape.com/Computing_and_Internet/Shareware/PC/Business/Finance/index.html - 47548 bytes, fetched on Wed May 3 13:33:18 2000
THE REGISTER: Trial focus shifts to Gates (Weight 20)
Posted 31/10/98 8:33am by Graham Lea Trial focus shifts to Gates Bill Gates will be appearing on videotape next week in the Washing...
http://www.theregister.co.uk/981031-000001.html - 12680 bytes, fetched on Wed May 3 13:04:26 2000
THE REGISTER: Lawyers spin after Gates video (Weight 20)
Posted 03/11/98 9:24am by Graham Lea Lawyers spin after Gates video The cordon sanitaire that has surrounded Bill Gates and prevent...
http://www.theregister.co.uk/981103-000003.html - 13722 bytes, fetched on Wed May 3 13:04:45 2000
B (Weight 19)
ADVERTISEMENT Go to: Fun & Games - Translator - Word of the Day - Help Search: the Web Dictionary Thesaurus for Web Di...
http://www.dictionary.com/Dir/Reference/Encyclopedia/Infoplease.com/Biographies/B/index.html - 20830 bytes, fetched on Wed May 3 13:09:12 2000
Angst (Weight 19)
Select from listnme.com TicketshopNewsReviewsSingles ArchivePick Of The MonthGigsChartsTop 50 of 19991998's Top 50 Albums1998's Top...
http://www.nme.com/board/angst/lofi.html - 77701 bytes, fetched on Wed May 3 14:14:09 2000
THE REGISTER: DoJ subpoenas Gates' former girlfriend over Vobis (Weight 18)
Posted 09/11/98 1:33pm by Graham Lea DoJ subpoenas Gates' former girlfriend over Vobis Stefanie Reichel, a former Microsoft manager...
http://www.theregister.co.uk/981109-000013.html - 11954 bytes, fetched on Wed May 3 13:03:29 2000
THE REGISTER: 'Cooked' MS memo - could Gates be in contempt of (Weight 18)
Posted 17/06/99 10:53am by John Lettice 'Cooked' MS memo - could Gates be in contempt of court? A highly dubious piece of spinning ...
http://www.theregister.co.uk/990617-000005.html - 12433 bytes, fetched on Wed May 3 13:03:55 2000
THE REGISTER: Gates thanks Compaq witness for help over trial (Weight 18)
Posted 24/02/99 4:51pm by Graham Lea Gates thanks Compaq witness for help over trial David Boies for the DoJ slowly set up Compaq V...
http://www.theregister.co.uk/990224-000023.html - 10455 bytes, fetched on Wed May 3 13:04:13 2000
THE REGISTER: Gates didn't threaten Intel's Grove, says Microso (Weight 18)
Posted 21/10/98 1:20pm by Graham Lea Gates didn't threaten Intel's Grove, says Microsoft attorney Bill Gates did not threaten Andy ...
http://www.theregister.co.uk/981021-000009.html - 10795 bytes, fetched on Wed May 3 13:04:30 2000
Personal Finance (Weight 17)
ADVERTISEMENT Go to: Fun & Games - Translator - Word of the Day - Help Search: the Web Dictionary Thesaurus for Web Di...
http://www.dictionary.com/Dir/Computers/Internet/WWW/Web_Portals/Netscape.com/Computing_and_Internet/Shareware/PC/Home_&_Personal/Personal_Finance/index.html□□ - 43750 bytes, fetched on Wed May 3 13:33:40 2000
THE REGISTER: Microsoft leaks its own memos (Weight 16)
Posted 12/10/98 9:24am by John Lettice Microsoft leaks its own memos Previous Bill Gates memos have reached the outside world via l...
http://www.theregister.co.uk/981012-000005.html - 12412 bytes, fetched on Wed May 3 13:04:32 2000
THE REGISTER: No surrender: Gates draws a line in the sand (Weight 16)
MONDAY MAY 1ST 2000 Full Register Comdex Fall Coverage The Register Bulletin Board Microsoft on Trial Useful Links The Quick Guide ...
http://www.theregister.co.uk/991111-000007.html - 15348 bytes, fetched on Wed May 3 13:03:37 2000
THE REGISTER: Gates: How Lotus boss helped MS inside IBM (Weight 16)
Posted 18/12/98 5:06pm by John Lettice Gates: How Lotus boss helped MS inside IBM Bill Gates' video testimony earlier this week inc...
http://www.theregister.co.uk/981218-000016.html - 12322 bytes, fetched on Wed May 3 13:04:47 2000

(matches continur for several pages…)

E.2 Server log output for multiple operations

The following shows server start-up and the processing of a few queries :

```
9671 Wed May 10 14:25:20 2000 - Search engine data server (May  3 2000 17:40:32)
9671 Wed May 10 14:25:20 2000 - Opening database...
9671 Wed May 10 14:25:20 2000 - Database at /home/thowat/project/fetchdb opened OK
9671 Wed May 10 14:25:20 2000 - Binding socket...
9671 Wed May 10 14:25:20 2000 - Calling listen...
9671 Wed May 10 14:25:20 2000 - Server listening on port 1234...
9752 Wed May 10 14:28:57 2000 - New connection recieved from mipc-21, socket 0xa
9752 Wed May 10 14:28:57 2000 - Transfer type 0x67 requested, data length 0x24
9752 Wed May 10 14:28:57 2000 - SNI_QUERY:
9752 Wed May 10 14:28:57 2000 - 1 query terms
9752 Wed May 10 14:28:57 2000 - sending SNI_QUERYRESULTS
9752 Wed May 10 14:28:57 2000 - transfer ok, closing session
9755 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9755 Wed May 10 14:28:58 2000 - Transfer type 0x67 requested, data length 0x24
9755 Wed May 10 14:28:58 2000 - SNI_QUERY:
9755 Wed May 10 14:28:58 2000 - 1 query terms
9755 Wed May 10 14:28:58 2000 - sending SNI_QUERYRESULTS
9755 Wed May 10 14:28:58 2000 - transfer ok, closing session
9756 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9756 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9756 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9756 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x46e0
9756 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD
9756 Wed May 10 14:28:58 2000 - transfer ok, closing session
9757 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9757 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9757 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9757 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x5a57
9757 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD
9757 Wed May 10 14:28:58 2000 - transfer ok, closing session
9759 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9759 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9759 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9759 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x1e2
9759 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD
9759 Wed May 10 14:28:58 2000 - transfer ok, closing session
9760 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9760 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9760 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9760 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x4ace
9760 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD
9760 Wed May 10 14:28:58 2000 - transfer ok, closing session
9761 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9761 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9761 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9761 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x740c
9761 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD
9761 Wed May 10 14:28:58 2000 - transfer ok, closing session
9763 Wed May 10 14:28:58 2000 - New connection recieved from mipc-21, socket 0xa
9763 Wed May 10 14:28:58 2000 - Transfer type 0x66 requested, data length 0x4
9763 Wed May 10 14:28:58 2000 - SNI_GETDOCCHUNK:
9763 Wed May 10 14:28:58 2000 - Request for DocRecord chunk 0x46e1
9763 Wed May 10 14:28:58 2000 - sending SNI_DOCRECORD

[...]
```

The following shows document submissions, queries and document record requests occurring.

```
10093 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10093 Wed May 10 14:29:12 2000 - Transfer type 0x64 requested, data length 0x42f
10093 Wed May 10 14:29:12 2000 - SNI_DOCSUBMISSION:
10090 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10090 Wed May 10 14:29:12 2000 - transfer ok, closing session
10091 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10091 Wed May 10 14:29:12 2000 - transfer ok, closing session
10092 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10092 Wed May 10 14:29:12 2000 - transfer ok, closing session
10094 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10094 Wed May 10 14:29:12 2000 - Transfer type 0x66 requested, data length 0x4
10094 Wed May 10 14:29:12 2000 - SNI_GETDOCCHUNK:
10094 Wed May 10 14:29:12 2000 - Request for DocRecord chunk 0x9e
10095 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10095 Wed May 10 14:29:12 2000 - Transfer type 0x66 requested, data length 0x4
10095 Wed May 10 14:29:12 2000 - SNI_GETDOCCHUNK:
10095 Wed May 10 14:29:12 2000 - Request for DocRecord chunk 0x403c
10094 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10094 Wed May 10 14:29:12 2000 - transfer ok, closing session
10096 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10096 Wed May 10 14:29:12 2000 - Transfer type 0x66 requested, data length 0x4
10096 Wed May 10 14:29:12 2000 - SNI_GETDOCCHUNK:
10096 Wed May 10 14:29:12 2000 - Request for DocRecord chunk 0x2953
10095 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10093 Wed May 10 14:29:12 2000 - SNI_OK: Document submitted OK
10093 Wed May 10 14:29:12 2000 - transfer ok, closing session
10096 Wed May 10 14:29:12 2000 - sending SNI_DOCRECORD
10098 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10098 Wed May 10 14:29:12 2000 - Transfer type 0x64 requested, data length 0x3dd
10098 Wed May 10 14:29:12 2000 - SNI_DOCSUBMISSION:
10100 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10100 Wed May 10 14:29:12 2000 - Transfer type 0x67 requested, data length 0x24
```

```
10100 Wed May 10 14:29:12 2000 - SNI_QUERY:
10100 Wed May 10 14:29:12 2000 - 1 query terms
10100 Wed May 10 14:29:12 2000 - sending SNI_QUERYRESULTS
10100 Wed May 10 14:29:12 2000 - transfer ok, closing session
10098 Wed May 10 14:29:12 2000 - SNI_OK: Document submitted OK
10098 Wed May 10 14:29:12 2000 - transfer ok, closing session
10103 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10103 Wed May 10 14:29:12 2000 - Transfer type 0x64 requested, data length 0x430
10103 Wed May 10 14:29:12 2000 - SNI_DOCSUBMISSION:
10103 Wed May 10 14:29:12 2000 - SNI_OK: Document submitted OK
10103 Wed May 10 14:29:12 2000 - transfer ok, closing session
10107 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10107 Wed May 10 14:29:12 2000 - Transfer type 0x64 requested, data length 0x36b
```

Note that in the bold section two sessions execute concurrently – their log messages are interleaved.

```
10108 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10108 Wed May 10 14:29:12 2000 - Transfer type 0x67 requested, data length 0x24
10107 Wed May 10 14:29:12 2000 - SNI_DOCSUBMISSION:
10108 Wed May 10 14:29:12 2000 - SNI_QUERY:
10108 Wed May 10 14:29:12 2000 - 1 query terms
10108 Wed May 10 14:29:12 2000 - sending SNI_QUERYRESULTS
10108 Wed May 10 14:29:12 2000 - transfer ok, closing session
10110 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10110 Wed May 10 14:29:12 2000 - Transfer type 0x67 requested, data length 0x24
10110 Wed May 10 14:29:12 2000 - SNI_QUERY:
10110 Wed May 10 14:29:12 2000 - 1 query terms
10110 Wed May 10 14:29:12 2000 - sending SNI_QUERYRESULTS
10110 Wed May 10 14:29:12 2000 - transfer ok, closing session
10107 Wed May 10 14:29:12 2000 - SNI_OK: Document submitted OK
10107 Wed May 10 14:29:12 2000 - transfer ok, closing session
10113 Wed May 10 14:29:12 2000 - New connection recieved from mipc-21, socket 0xa
10113 Wed May 10 14:29:12 2000 - Transfer type 0x64 requested, data length 0x2c5
10113 Wed May 10 14:29:12 2000 - SNI_DOCSUBMISSION:
10115 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10115 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10115 Wed May 10 14:29:13 2000 - SNI_QUERY:
10115 Wed May 10 14:29:13 2000 - 1 query terms
10115 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10115 Wed May 10 14:29:13 2000 - transfer ok, closing session
10113 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10113 Wed May 10 14:29:13 2000 - transfer ok, closing session
10118 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10118 Wed May 10 14:29:13 2000 - Transfer type 0x64 requested, data length 0x42f
10118 Wed May 10 14:29:13 2000 - SNI_DOCSUBMISSION:
10118 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10118 Wed May 10 14:29:13 2000 - transfer ok, closing session
10121 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10121 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10121 Wed May 10 14:29:13 2000 - SNI_QUERY:
10121 Wed May 10 14:29:13 2000 - 1 query terms
10121 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10121 Wed May 10 14:29:13 2000 - transfer ok, closing session
10123 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10123 Wed May 10 14:29:13 2000 - Transfer type 0x64 requested, data length 0x42f
10123 Wed May 10 14:29:13 2000 - SNI_DOCSUBMISSION:
10125 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10125 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10125 Wed May 10 14:29:13 2000 - SNI_QUERY:
10125 Wed May 10 14:29:13 2000 - 1 query terms
10125 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10125 Wed May 10 14:29:13 2000 - transfer ok, closing session
10123 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10123 Wed May 10 14:29:13 2000 - transfer ok, closing session
10128 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10128 Wed May 10 14:29:13 2000 - Transfer type 0x64 requested, data length 0x42f
10128 Wed May 10 14:29:13 2000 - SNI_DOCSUBMISSION:
10129 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10129 Wed May 10 14:29:13 2000 - Transfer type 0x66 requested, data length 0x4
10129 Wed May 10 14:29:13 2000 - SNI_GETDOCCHUNK:
10129 Wed May 10 14:29:13 2000 - Request for DocRecord chunk 0x2db1
10129 Wed May 10 14:29:13 2000 - sending SNI_DOCRECORD
10129 Wed May 10 14:29:13 2000 - transfer ok, closing session
10131 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10131 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10131 Wed May 10 14:29:13 2000 - SNI_QUERY:
10131 Wed May 10 14:29:13 2000 - 1 query terms
10131 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10131 Wed May 10 14:29:13 2000 - transfer ok, closing session
10132 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10132 Wed May 10 14:29:13 2000 - Transfer type 0x66 requested, data length 0x4
10132 Wed May 10 14:29:13 2000 - SNI_GETDOCCHUNK:
10132 Wed May 10 14:29:13 2000 - Request for DocRecord chunk 0x2db1
10128 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10128 Wed May 10 14:29:13 2000 - transfer ok, closing session
10132 Wed May 10 14:29:13 2000 - sending SNI_DOCRECORD
10132 Wed May 10 14:29:13 2000 - transfer ok, closing session
10135 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10135 Wed May 10 14:29:13 2000 - Transfer type 0x64 requested, data length 0x278
10135 Wed May 10 14:29:13 2000 - SNI_DOCSUBMISSION:
10135 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10135 Wed May 10 14:29:13 2000 - transfer ok, closing session
10138 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10138 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10138 Wed May 10 14:29:13 2000 - SNI_QUERY:
10138 Wed May 10 14:29:13 2000 - 1 query terms
10138 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10138 Wed May 10 14:29:13 2000 - transfer ok, closing session
10140 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10140 Wed May 10 14:29:13 2000 - Transfer type 0x66 requested, data length 0x4
10140 Wed May 10 14:29:13 2000 - SNI_GETDOCCHUNK:
10140 Wed May 10 14:29:13 2000 - Request for DocRecord chunk 0x2db1
10140 Wed May 10 14:29:13 2000 - sending SNI_DOCRECORD
10140 Wed May 10 14:29:13 2000 - transfer ok, closing session
10141 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
```

```
10141 Wed May 10 14:29:13 2000 - Transfer type 0x64 requested, data length 0x42f
10141 Wed May 10 14:29:13 2000 - SNI_DOCSUBMISSION:
10143 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10143 Wed May 10 14:29:13 2000 - Transfer type 0x67 requested, data length 0x24
10143 Wed May 10 14:29:13 2000 - SNI_QUERY:
10143 Wed May 10 14:29:13 2000 - 1 query terms
10143 Wed May 10 14:29:13 2000 - sending SNI_QUERYRESULTS
10143 Wed May 10 14:29:13 2000 - transfer ok, closing session
10145 Wed May 10 14:29:13 2000 - New connection recieved from mipc-21, socket 0xa
10145 Wed May 10 14:29:13 2000 - Transfer type 0x66 requested, data length 0x4
10145 Wed May 10 14:29:13 2000 - SNI_GETDOCCHUNK:
10145 Wed May 10 14:29:13 2000 - Request for DocRecord chunk 0x2db1
10141 Wed May 10 14:29:13 2000 - SNI_OK: Document submitted OK
10141 Wed May 10 14:29:13 2000 - transfer ok, closing session
10145 Wed May 10 14:29:13 2000 - sending SNI_DOCRECORD
10145 Wed May 10 14:29:13 2000 - transfer ok, closing session
```

E.3 HTML Preprocessor Output

This is for a sample document downloaded from http://www.isotech.co.uk/imdex.html - note that it contains bad HTML which the parser copes with. The hex figures are word CRCs.

```
 preprochtml (May  3 2000 17:40:28)
/export/wgot/www.isotech.co.uk/index.html
processing...
no following tag found
quicksorting..
   0 14               calibr b4b202f2
   1  7           temperatur 5c4b652b
   2  6                equip  c77f297
   3  3             isotherm ab13f973
   4  3            technolog 290779e7
   5  3               servic bb959935
   6  5                  lab c0559af5
   7  5              english 1eb0c8b0
   8  4              product 8709f629
   9  4          thermocoupl 9b787478
  10  3                block 41717f51
  11  3                 bath 639231d2
  12  3               furnac c3a1a781
  13  2                  dry 5beec62e
  14  2              portabl f382a3c2
  15  2                  fix 632667e6
  16  2                point 55b68376
  17  2              primari c866a4a6
  18  2             referenc ca63b603
  19  2                  ltd d41506b5
  20  2               involv 7eee432e
  21  2               includ a6843604
  22  2                   us 7258b958
  23  2                 page 12904090
  24  2               inform 624fdbaf
  25  2                refer 29333f53
  26  2                 here c0f2feb2
  27  2               articl cf903d90
  28  2              isotech 40a72447
  29  2                  new e6657ac5
  30  2                 tour 2e4796a7
  31  2                 help df1a9c5a
  32  2                 http c4b362d3
  33  2            copyright 2c2f59cf
  34  1                choos b905c9a5
  35  1                 then 48d9c3f9
  36  1                   go 61740ff4
  37  1                 true e123f543
  38  1                indic 6e74e0f4
  39  1             apparatu 87b251f2
  40  1                 isoc 99ac6c2c
  41  1               liquid 69b27ff2
  42  1              fluidis  5fd6a5d
  43  1                autom  a2aef6a
  44  1              softwar f313a173
  45  1             handheld 87771997
  46  1                black ae506450
  47  1                 bodi 9da13981
  48  1                sourc caae7f6e
  49  1             accredit 2af34493
  50  1             research 5d196e39
  51  1              develop 2bd8d0d8
  52  1           manufactur f94e308e
  53  1               produc bd877d67
  54  1             standard 7e138c73
  55  1             platinum be8dbf2d
  56  1               resist  cbd4b1d
  57  1            thermomet 913b065b
  58  1                 cell  ef851f8
  59  1               associ  dc924e9
  60  1             metrolog c07a6f3a
  61  1           laboratori 169955b9
  62  1           throughout ad967f56
  63  1                world 73032363
  64  1             industri 775e7c9e
  65  1               system 9053aa33
  66  1               provid 913d669d
  67  1                  our e6431e23
  68  1                 rang 7b623722
  69  1                along 10332653
  70  1               materi 42ddc97d
  71  1             interest 194a900a
  72  1               measur 12772c97
  73  1                metal 75c84cc8
  74  1                  rtd 28375ed7
  75  1                 high 19f8a6f8
  76  1             accuraci 6b2aae6a
  77  1           comparison 7e283b28
  78  1                  can 3e487748
  79  1                found 75da6e9a
  80  1              journal 65d85ed8
  81  1                 line d0bc073c
  82  1                  prt  c208820
  83  1               calcul  928ac28
  84  1                 data c3228762
  85  1                evalu ce699349
  86  1               report 3b505150
  87  1                gener  21fb1ff
  88  1              contact 23026342
  89  1                 find 31767fb6
  90  1              exhibit 5e3d499d
  91  1                 site d96d24cd
  92  1                updat 71e73d07
  93  1               public d5cacc8a
  94  1                  uka 414f98af
  95  1              schedul a4502e50
  96  1            secondari 74e2c8a2
  97  1               search a6ee9b2e
  98  1                  map d1769fb6
  99  1                  get 7ed7d437
 100  1             catalogu b953a333
 101  1                train 789cab1c
 102  1                cours 8da56905
 103  1                 book 9619c539
 104  1                email 199cea1c
 105  1             newslett 76683b68
 106  1              version 48972a77
 107  1              deutsch a5ffea1f
 108  1        klasmeier.com 46b90199
 109  1                other f4416c61
 110  1              languag eee032e0
 111  1                babel ea27dec7
 112  1                 fish 2db5cb15
 113  1                below cf6643a6
 114  1                 type 5ea7fa47
 115  1                 past  7d78d37
 116  1                 text 887c47fc
 117  1                  web 6dc3e5a3
 118  1              address 42600e60
 119  1                begin 73c99ae9
 120  1    www.isotech.co.uk c1c968e9
 121  1                power a2286728
 122  1              systran 3b962956
 123  1             translat 2604a684
 124  1               from:  ae8e804e
 125  1               french df165dd6
 126  1               german c6466e86
 127  1              italian 43efce0f
 128  1            portugues 2783e7e3
 129  1              spanish b0a4e424
 130  1   info@isotech.co.uk 7db95a99
 131  1              mainten 671f74ff
0.020000 seconds to process 9743 bytes, 487150.000000
bytes/sec
Document title    = Temperature Calibration from
Isothermal Technology
Document abstract = Temperature Calibration Equipment
and Services Products Choose a Product then Go! Dry
Block Baths Portable Temperature Calibrators
exiting OK, 0
```

APPENDIX F – SOURCE CODE

F.1 cbtree.c

```
/*
 * cbtree.c
 *
 * Implements a binary tree indexed chunked tree
 *
 * ** use polynominal hash so there is a 1:1 mapping of hash->key. **
 * entries for keys are never deleted, just set to -1
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"
#include "chunks.h"

//#define CBTREETEST 1

typedef struct _cbtfile {
  FILE *index;      /* the index (btree) file */
  chunkfile *main;  /* chunkfile - ie the main data */
  char *filen;      /* filename of main file (for debugging) */
} cbtfile;

typedef struct _cbindex {
  int        hash;  /* the hash value this list is for */
  chunk_id   left;  /* left in tree (chunk_id refers to index file) */
  chunk_id   right; /* right in tree (chunk_id refers to index file) */
  chunk_id   mfchunk; /* chunk in main file */
} cbindex;

/* open a chunked btree file
 */
cbtfile *cbtree_open(char *file, int chunksize)
{
  cbtfile *f;
  char *filei;

  f=malloc(sizeof(cbtfile));

  if(f==NULL)
  {
    fprintf(stderr,"cbtfile: failed to allocate %i bytes for cbtfile\n",
            sizeof(cbtfile));
    return NULL;
  }

  filei=malloc(strlen(file)+7);
  if(filei==NULL)
  {
    free(f);
    fprintf(stderr,"cbtfile: failed to allocate %i bytes for filei\n",
            strlen(file)+7);
    return NULL;
  }

  sprintf(filei,"%s.index",file);

  f->filen=malloc(strlen(file)+1);
  if(f->filen==NULL)
  {
    free(f);
    free(filei);
    fprintf(stderr,"cbtfile: failed to allocate %i bytes for f->filen\n",
            strlen(file)+1);
    return NULL;
  }

  strcpy(f->filen,file);

  if((f->index=fopen(filei,"rb+")) == NULL)
    f->index=fopen(filei,"wb+");

  if(f->index==NULL)
  {
    fprintf(stderr,"cbtree_open: failed to open %s\n",index);
    return NULL;
  }

  setbuf(f->index,NULL);
  rewind(f->index);

  f->main=chunk_open(file,chunksize);
  free(filei);

  if((f->index == NULL) | (f->main == NULL))
  {
    free(f);
    fprintf(stderr,"cbtfile: failed to open main for %s\n",file);
    return NULL;
  }

  return(f);
}

/* close a chunked btree file
 */
void cbtree_close(cbtfile *cf)
{
  fclose(cf->index);
  chunk_close(cf->main);
  free(cf->filen);
  free(cf);
}

/* find the number of individual chunk chains in a file */
int cbtree_countentries(cbtfile *cf)
{
  long tmp, len;

  if(lock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_countentries: failed to lock %s",cf->filen);
    return -1;
  }

  tmp=ftell(cf->index);
  fseek(cf->index, 0, SEEK_END);
  len=ftell(cf->index);
  fseek(cf->index, tmp, SEEK_SET);

  if(unlock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_countentries: failed to unlock %s",cf->filen);
    return -1;
  }

  return((int)(len/sizeof(cbindex)));
}

static int read_cbindex(cbtfile *cf, cbindex *cbi, int pos)
{
  fseek(cf->index, pos*sizeof(cbindex), SEEK_SET);
  return(fread(cbi,1,sizeof(cbindex),cf->index));
}

static int write_cbindex(cbtfile *cf, cbindex *cbi, int pos)
{
  fseek(cf->index, pos*sizeof(cbindex), SEEK_SET);
  return(fwrite(cbi,1,sizeof(cbindex),cf->index));
}

/* returns chunk pointer for a cbindex entry
 */
chunk_id cbtree_getchunk(cbtfile *cf, int index)
{
  cbindex cbi;

  read_cbindex(cf,&cbi,index);
  return cbi.mfchunk;
}

/* just locate a chunk in cf (returns an integer index for cbindex
 * record for that hash) - returns -1 if no such entry in the index
 * exists. returned figure should be passed to future operations on
 * this hash - it is an indexpoint.
 */
int cbtree_locate(cbtfile *cf, int hash)
{
  cbindex data;
  int length;
  int chnk=0;

#ifdef DEBUG
  printf("cbtree_locate(0x%x,0x%x)\n",cf,hash);
#endif

  if(lock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_locate: failed to lock %s",cf->filen);
    return -1;
  }

  while(chnk!=-1)
  {
    if(!read_cbindex(cf, &data, chnk))
    {
      if(chnk!=0) /* could be an empty file */
        fprintf(stderr,"cbtree_locate: Failed to retrieve index chunk %i in %s\n",chnk,cf->filen);
      if(unlock_file(cf->index) == -1)
        fprintf(stderr,"cbtree_locate: failed to unlock %s",cf->filen);
      return -1;
    }

#ifdef DEBUG
    printf("(%i) hash = 0x%x, left = %i, right = %i\n",chnk, data.hash,data.left, data.right);
#endif

    if(data.hash == hash)
    {
      /* we have it */
      if(unlock_file(cf->index) == -1)
      {
        fprintf(stderr,"cbtree_locate: failed to unlock %s",cf->filen);
        return -1;
      }
      return chnk;
    } else {
      if(data.hash > hash)
      {
        chnk=data.left;
        //printf("left -> %i ",chnk);
      } else {
        chnk=data.right;
        //printf("right -> %i ",chnk);
      }
    }
  }

  if(unlock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_locate: failed to unlock %s",cf->filen);
    return -1;
  }
  return -1;
}

/* retrieve a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns chunk_id of main file containing matching record
 */
chunk_id cbtree_retrieve(cbtfile *cf, int hash, char **data, int *length, int ichnk)
{
  cbindex cbi;
  char *cdata;
  int clength;

  if(ichnk==-1)
    ichnk=cbtree_locate(cf,hash);  /* find location in index */

  if(lock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_retrieve: failed to lock %s",cf->filen);
    return -1;
  }

  if(ichnk==-1)
  {
    fprintf(stderr,"cbtree_retrieve: failed to locate tree entry for "
                   "hash 0x%x in %s index\n",hash,cf->filen);
    if(unlock_file(cf->index) == -1)
    {
      fprintf(stderr,"cbtree_retrieve: failed to unlock %s",cf->filen);
      return -1;
    }
    return -1;
  }

  read_cbindex(cf,&cbi,ichnk);    /* read index information */

  /* unlock, we don't need index any more */
  if(unlock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_retrieve: failed to unlock %s",cf->filen);
    return -1;
  }

  if(chunk_retrieve(cf->main,cbi.mfchunk,&cdata,&clength))
  {
    fprintf(stderr,"cbtree_retrieve: Failed to retrieve data"
                   " chunk %i\n",cbi.mfchunk);
    return -1;
  }

  *data=cdata;
  *length=clength;
  return cbi.mfchunk;
}

/* append to a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns 0 if successful,
 * non-zero if there was a problem...
 */

int cbtree_append(cbtfile *cf, int hash, char *data, int length, int ichnk)
{
  cbindex cbi;
  char *cdata;
  int clength;

#ifdef DEBUG
  printf("cbtree_append(0x%x,0x%x,0x%x,%i,%i)\n",(int)cf,hash,(int)data,(int)length,ichnk);
#endif

  if(ichnk==-1)
    ichnk=cbtree_locate(cf,hash);  /* find location in index */

  if(ichnk==-1)
  {
    fprintf(stderr,"cbtree_append: failed to locate tree entry for hash 0x%x in %s.index\n",hash,cf->filen);
    return -1;
  }

  /* lock the index whilst we retrieve */
  if(lock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_append: failed to lock %s",cf->filen);
    return -1;
  }

  read_cbindex(cf,&cbi,ichnk);    /* read index information */

  if(unlock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_append: failed to unlock %s",cf->filen);
    return -1;
  }

  if(chunk_retrieve(cf->main,cbi.mfchunk,&cdata,&clength))
  {
    fprintf(stderr,"cbtree_retrieve: Failed to retrieve data"
                   " chunk %i\n",cbi.mfchunk);
    return -1;
  }

  return(chunk_append(cf->main, cbi.mfchunk, data, length));
}

chunkfile *cbtree_getchunkfilehandle(cbtfile *cf)
{
  return cf->main;
}

/* add a new hash to the index and create chunk
 */
chunk_id cbtree_allocate(cbtfile *cf, int hash, char *data, int length)
{
  int chnk=0,lastchnk=0;
  chunk_id cid;
  cbindex cbi;
  int curloc=0;

#ifdef DEBUG
  printf("cbtree_allocate(0x%x,0x%x,0x%x,%i)\n",(int)cf,hash,(int)data,length);
#endif

  /* first allocate the normal chunk */
  cid=chunk_allocate(cf->main,data,length);

  if(cid==-1)
  {
    fprintf(stderr,"cbtree_allocate: chunk_allocate call failed us!\n");
    return(cid);
  }

  /* now lock the index */
  if(lock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_allocate: failed to lock %s",cf->filen);
    return -1;
  }

  /* cid now points to the head of the chunk chain we have just
   * created, now to create an entry in the index for it
   */

  while(chnk!=-1)
  {
    if(!read_cbindex(cf, &cbi, chnk))
    {
      if(chnk==0) /* if tree was empty we expect to fail to read entry */
      {
        /* write it in the first position */
        cbi.left=-1;
        cbi.right=-1;
        cbi.hash=hash;
        cbi.mfchunk=cid;
        if(!write_cbindex(cf,&cbi,chnk))
        {
          fprintf(stderr,"cbtree_allocate: Failed to write index chunk %i in file %s\n",chnk,cf->filen);
          if(unlock_file(cf->index) == -1)
          {
            fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
            return -1;
          }
          chunk_deallocate(cf->main,cid);
          return -1;
        }
        return(cid);
      } else {
        fprintf(stderr,"cbtree_allocate: Failed to retrieve index chunk %i in file %s\n",chnk,cf->filen);
        if(unlock_file(cf->index) == -1)
        {
          fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
          return -1;
        }
        chunk_deallocate(cf->main,cid);
        return -1;
      }
    }

    if(cbi.hash == hash)
    {
      /* we have it */
      if(cbi.mfchunk!=-1)
      {
        fprintf(stderr,"cbtree_allocate: SERIOUS! tried to allocate tree "
                       "entry for hash 0x%x, but it already exists in "
                       "%s.index! (currently pointing at chunk_id 0x%x)",
                       hash,cf->filen,cbi.mfchunk);
        chunk_deallocate(cf->main,cid);
        if(unlock_file(cf->index) == -1)
        {
          fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
          return -1;
        }
        return -1;
      } else {
        /* lastchunk will point to the empty tree node. just write over
         * it
         */
        cbi.mfchunk=cid;
        if(!write_cbindex(cf,&cbi,lastchnk))
        {
          fprintf(stderr,"cbtree_allocate: Failed to write index chunk %i in file %s\n",chnk,cf->filen);
          chunk_deallocate(cf->main,cid);
          if(unlock_file(cf->index) == -1)
          {
            fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
            return -1;
          }
          return -1;
        }
      }
    } else {
      if(cbi.hash > hash)
      {
        lastchnk=chnk;
        chnk=cbi.left;
      } else {
        lastchnk=chnk;
        chnk=cbi.right;
      }
    }
  }

  fseek(cf->index,0,SEEK_END);
  curloc=ftell(cf->index);

  /* new tree node at eof, link into last chunk */
  cbi.left=-1;
  cbi.right=-1;
  cbi.hash=hash;
  cbi.mfchunk=cid;
  if(!write_cbindex(cf,&cbi,(curloc/sizeof(cbindex))))
  {
    fprintf(stderr,"cbtree_allocate: Failed to write index chunk %i in file %s\n",(curloc/sizeof(cbindex)),cf-
>filen);
    chunk_deallocate(cf->main,cid);
    if(unlock_file(cf->index) == -1)
    {
      fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
      return -1;
    }
    return -1;
  }

  /* load previous node */
  if(!read_cbindex(cf, &cbi, lastchnk))
  {
    fprintf(stderr,"cbtree_allocate: Failed to retrieve index chunk %i in file %s\n",lastchnk,cf->filen);
    chunk_deallocate(cf->main,cid);
    if(unlock_file(cf->index) == -1)
    {
      fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
      return -1;
    }
    return -1;
  }

  if(cbi.hash > hash)
  {
    if(cbi.left!=-1)
    {
      fprintf(stderr,"cbtree_allocate: SERIOUS left pointer of index entry 0x%x in file %s.index is already set to
0x%x!\n",lastchnk,cf->filen,cbi.left);
      chunk_deallocate(cf->main,cid);
      if(unlock_file(cf->index) == -1)
      {
        fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
        return -1;
      }
      return -1;
    }
    cbi.left=curloc/sizeof(cbindex);
  } else {
    if(cbi.right!=-1)
    {
      fprintf(stderr,"cbtree_allocate: SERIOUS right pointer of index entry 0x%x in file %s.index is already set to
0x%x!\n",lastchnk,cf->filen,cbi.right);
      chunk_deallocate(cf->main,cid);
      if(unlock_file(cf->index) == -1)
      {
        fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
        return -1;
      }
      return -1;
    }
    cbi.right=curloc/sizeof(cbindex);
  }

  /* write new version of parent node */
  if(!write_cbindex(cf, &cbi, lastchnk))
  {
    fprintf(stderr,"cbtree_allocate: Failed to write index chunk %i in file %s\n",lastchnk,cf->filen);
    chunk_deallocate(cf->main,cid);
    if(unlock_file(cf->index) == -1)
    {
      fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
      return -1;
    }
    return -1;
  }

  if(unlock_file(cf->index) == -1)
  {
    fprintf(stderr,"cbtree_allocate: failed to unlock %s",cf->filen);
    return -1;
  }

  return(cid);
}

#ifdef CBTREETEST
/* the main program */
int main(void)
{
  FILE *dict;
  char inbuffer[255];
  cbtfile *cbt;
  chunk_id moo;
  char *cdata;
  int length;
  char richard[]="Richard";
  int i=10;

  setbuf(stdout,NULL);

  dict=fopen("/usr/dict/words","r");

  if(!dict)
  {
    fprintf(stderr,"Failed to open dictionary\n");
    exit(1);
  }

  cbt=cbtree_open("/home/thowat/project/cbtest",4);

  while(!feof(dict))
  {
    chunk_id cid;
    int indexpoint;

    --i;

    fgets(inbuffer,255,dict);
    if((indexpoint=cbtree_locate(cbt,expohash(inbuffer,strlen(inbuffer)))) != -1)
    {
      printf("MATCHED chunk_locate indexpoint = %i for %s\n",indexpoint,inbuffer);
    } else {
      printf("ALLOCATING for %s\n",inbuffer);
      cid=cbtree_allocate(cbt, expohash(inbuffer,strlen(inbuffer)), inbuffer, strlen(inbuffer));
      printf("cid %i string = %s",cid,inbuffer);
    }
  }

  fclose(dict);

  /* expohash of Whiteley */
  moo=cbtree_retrieve(cbt,0x14de5173,&cdata,&length,-1);
  cdata[length]='\0';
  printf("PIG");
  printf("chunk_id=0x%x data=%s\n",moo,cdata);
  printf("COW");

  /* test append */
  cbtree_append(cbt, 0x14de5173, richard, strlen(richard)+1,-1);

  /* expohash of Whiteley */
  moo=cbtree_retrieve(cbt,0x14de5173,&cdata,&length,-1);
  printf("chunk_id=0x%x data=%s\n",moo,cdata);

  cbtree_close(cbt);
}

#endif
```

F.2 cbtree.h

```
/*
 * cbtree.h
 *
 * Implements a binary tree indexed chunked tree
 *
 * ** use polynominal hash so there is a 1:1 mapping of hash->key. **
 * entries for keys are never deleted, just set to -1
 */

typedef struct _cbtfile {
  FILE *index;      /* chunkfile - ie the main data */
  chunkfile *main;  /* the index (btree) file */
  char *filen;      /* filename of main file (for debugging) */
} cbtfile;
```

```
/* open a chunked btree file
 */
cbtfile *cbtree_open(char *file, int chunksize);

/* close a chunked btree file
 */
void cbtree_close(cbtfile *cf);

/* find the number of individual chunk chains in a file */
int cbtree_countentries(cbtfile *cf);

/* just locate a chunk in cf (returns an integer index for cbindex
 * record for that hash) - returns -1 if no such entry in the index
 * exists. returned figure should be passed to future operations on
 * this hash - it is an indexpoint.
 */
int cbtree_locate(cbtfile *cf, int hash);

/* retrieve a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns chunk_id of main file containing matching record
 */
chunk_id cbtree_retrieve(cbtfile *cf, int hash, char **data, int *length, int ichnk);

/* append to a chunk from main file matching hash
 * pass an indexpoint if one has been previously looked up by cbtree_locate
 * for this hash
 *
 * returns 0 if successful,
 * non-zero if there was a problem...
 */
int cbtree_append(cbtfile *cf, int hash, char *data, int length, int ichnk);

/* get chunk file handle */
chunkfile *cbtree_getchunkfilehandle(cbtfile *cf);

/* add a new hash to the index and create chunk
 */
chunk_id cbtree_allocate(cbtfile *cf, int hash, char *data, int length);

/* returns chunk pointer for a cbindex entry
 */
chunk_id cbtree_getchunk(cbtfile *cf, int index);
```

F.3 chunks.c

```
/*
 * chunks.c
 * Tony Howat 1999/2000
 *
 * This code administers blocks of data stored in files - these blocks
 * of data are split into "chunks" of uniform length, which are chained
 * together.
 *
 * This give us random-access type performance with variable length
 * records.
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"
#include "chunks.h"
#include "debug.h"

/* file format of a "chunked" file :
 *
 * 4 bytes ID String        : CHNK
 * 4 bytes First Free Chunk : offsets from next word, ie first chunk is
 *                            zero. Chunk offsets are in terms of chunks,
 *                            rather than bytes.
 * --
 *
 * and the chunks are :
 *
 * 4 bytes length           : ie, length of data in *this* chunk (-1 = empty)
 * 4 bytes next             : chunk offset to next chunk in this chain
 *                            (-1 = end of chunk chain)
 */

#ifdef SKIPTHIS

/* these definitions are in chunks.h */

typedef int chunk_id;

typedef struct _chunkfile {
  FILE *fp;          /* normal C file pointer for chunk file */
  int chunksize;     /* size of data area per chunk */
  char *filename;    /* filename for reference */
} chunkfile;

#endif /* SKIPTHIS */

typedef struct _chunkheader {
  int length;
  chunk_id next;
} chunkheader;

/* this little macro turns a chunk_id into a byte offset from the start of a
 * chunked file (cid=chunk_id clen=chunk length) */

#define CID_TO_BYTE(cid,clen) (cid*(clen+8))+(sizeof(chunkheader))

/* magic number for a chunk file */
#define CHNK 0x4b4e4843

/* open a chunked file
 */
chunkfile *chunk_open(char *file,int chunksize)
{
  FILE *fp;
  chunkfile *cf;

  if((fp=fopen(file,"rb+")) == NULL)
    fp=fopen(file,"wb+");

  if(fp==NULL)
  {
    fprintf(stderr,"chunk_open: failed to open %s\n",file);
    return NULL;
  }

  setbuf(fp,NULL); /* turn buffers off to ensure writes occur immediately */
  cf=(chunkfile *)malloc(sizeof(chunkfile));

  if(cf==NULL)
  {
    fprintf(stderr,"chunk_open: failed to allocate memory for %s\n",file);
    fclose(fp);
    return NULL;
  }

  cf->chunksize=chunksize;
  cf->fp=fp;
  cf->filename=(char *)malloc(strlen(file)+1);

  if(cf->filename==NULL)
  {
    fprintf(stderr,"chunk_open: failed to allocate memory for %s\n",file);
    fclose(fp);
    free(cf);
    return NULL;
  }

  strcpy(cf->filename,file);

  /* is the file empty? if so, write a header */
  fseek(cf->fp, 0, SEEK_END);
  if(ftell(cf->fp) < 8)
  {
    int tmp=-1;

    fwrite("CHNK", 4, 1, cf->fp); /* write the CHNK ID string */
    fwrite(&tmp, sizeof(int), 1, cf->fp); /* write the head of the free list */
  }

  {
    int tmp;

    rewind(cf->fp);
    fread(&tmp, sizeof(int), 1, cf->fp);  /* read ID string */

    if(tmp!=CHNK)
    {
      fprintf(stderr,"chunk_open: bad ID (0x%x) string in %s\n",tmp,file);
      fclose(fp);
      free(cf->filename);
      free(cf);
      return NULL;
    }
  }

  rewind(cf->fp);

#ifdef DEBUG
  printf("(chunkfile *)0x%x=chunk_open(\"%s\",(int chunksize)%i)\n",(int)cf,
         file,chunksize);
  printf("cf->fp=%x\n",(int)cf->fp);
#endif

  return cf;
}

/* close a chunked file
 */

void chunk_close(chunkfile *cf)
{
  fclose(cf->fp);
  free(cf->filename);
  free(cf);
}

/* this function is static so locks have already been claimed */
static chunk_id allocate_next_free_chunk(chunkfile *cf)
{
  int allocated, newfree;
  chunkheader ch;
  int c=0;

#ifdef DEBUG
```

```
  printf("Starting allocate_next_free_chunk((chunkfile *)0x%x)\n",(int)cf);
#endif

  fseek(cf->fp, 4, SEEK_SET);      /* jump to head of free list */
  fread(&allocated, sizeof(chunk_id), 1, cf->fp); /* load value */

#ifdef DEBUG
  printf("free list head = 0x%x\n",allocated);
#endif

  if(allocated==-1)
  {
    int tmp;

    fseek(cf->fp, 0, SEEK_END);

    tmp=(int)ftell(cf->fp);
    tmp=tmp-(2*sizeof(chunk_id));
    allocated=(tmp/(cf->chunksize + (2 * sizeof(chunk_id)))); /* +1 removed */

    if((tmp%(cf->chunksize + (2 * sizeof(chunk_id)))))
    {
      fprintf(stdout,"allocate_next_free_chunk: %s is corrupted, not aligned to chunk boundary, %i bytes over (chunk
size==%i, file pos-headersize==%i)! Exiting.\n",cf->filename,(tmp%(cf->chunksize + (2 * sizeof(chunk_id)))),cf-
>chunksize,tmp);
      fprintf(stderr,"allocate_next_free_chunk: %s is corrupted, not aligned to chunk boundary, %i bytes over (chunk
size==%i, file pos-headersize==%i)! Exiting.\n",cf->filename,(tmp%(cf->chunksize + (2 * sizeof(chunk_id)))),cf-
>chunksize,tmp);
      return -1;
    }

#ifdef DEBUG
    printf("ftell(cf->fp) = %i\n",(int)ftell(cf->fp));
    printf("tmp-(2*sizeof(chunk_id)) = %i (2x sizeof = %i)\n",tmp,(2*sizeof(chunk_id)));
    printf("tmp/(cf->chunksize) = %i (cf->chunksize = %i)\n",allocated,cf->chunksize);
    printf("%i length - %i ",(int)ftell(cf->fp),cf->chunksize);
#endif
    newfree=-1;
  } else {

    chunkheader tmp;

    /* now we have to find the next pointer from the chunk we have
     * just allocated, so we can alter the head of the free list  */

    fseek(cf->fp, CID_TO_BYTE(allocated, cf->chunksize), SEEK_SET);

    /* NB: fread/fwrite return the number of ITEMS written/read, not bytes
     * necessarily!
     */
    if(fread(&tmp, sizeof(chunkheader), 1, cf->fp) != 1)
      newfree=-1;    /* load value */
    else
      newfree=tmp.next;
#ifdef DEBUG
    printf("tmp.next = 0x%x\n",tmp.next);
#endif
  }

  fseek(cf->fp, 4, SEEK_SET);      /* jump to head of free list */
  fwrite(&newfree, sizeof(chunk_id), 1, cf->fp); /* write value */

  /* write a dummy chunk header for new chunk, and null data */
  fseek(cf->fp, CID_TO_BYTE(allocated, cf->chunksize), SEEK_SET);
  ch.next=-1;
  ch.length=0;
  fwrite(&ch, sizeof(chunkheader), 1, cf->fp);
  while(c!=cf->chunksize)
  {
    fputc('\0',cf->fp);
    ++c;
  }

#ifdef DEBUG
  printf("new free pointer 0x%x\n",newfree);
  printf("allocate_next_free_chunk returns 0x%x\n",allocated);
#endif

  return allocated;
}

chunk_id chunk_allocate(chunkfile *cf, char *data, int length)
{
  chunk_id firstchunk=-1;
  chunkheader ch;
  int aw=0;

#ifdef DEBUG
  printf("Starting chunk_allocate((chunkfile *)0x%x,(char *)0x%x,(int)%i)\n"
         ,(int)cf,(int)data,length);
  printf("cf->fp=%x\n",(int)cf->fp);
#endif

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_allocate: failed to lock %s",cf->filename);
    return -1;
  }

  /* 1) find next free chunk location, and if necessary claim it
   * 2) create chunk block, and save it out to the appropriate
   *    location...
   * 3) if we haven't written all our data, do another block
   */

  ch.next=-1;
  while(length)
  {
    int newchunk;

    if(ch.next==-1)
      newchunk=allocate_next_free_chunk(cf);
    else
      newchunk=ch.next;

    if(firstchunk==-1)
      firstchunk=newchunk;

    if(length>(cf->chunksize))
    {
      ch.length=cf->chunksize;
      ch.next=allocate_next_free_chunk(cf);
      //if(ch.next==newchunk)
      //  ch.next=newchunk+1;
    } else {
      ch.length=length;
      ch.next=-1;
    }

#ifdef DEBUG
    printf("newchunk = %x newchunk.next = %x\n",newchunk,ch.next);
#endif

    /* fseek, fwrite, etc */
    fseek(cf->fp, CID_TO_BYTE(newchunk,cf->chunksize), SEEK_SET);
#ifdef DEBUG
    printf("writing chunk 0x%x at %i in file, length %i bytes\n", newchunk, CID_TO_BYTE(newchunk,cf-
>chunksize),ch.length);
    printf("DATA: [");
    fwrite(data, ch.length, 1, stdout);
    printf("]");
#endif
    fwrite(&ch, sizeof(chunkheader), 1, cf->fp);
    aw=fwrite(data, sizeof(char), ch.length, cf->fp);

    if(aw<cf->chunksize)
    {
      while(aw!=cf->chunksize)
      {
        fputc('\0',cf->fp);
        ++aw;
      }
    }
    data=data+ch.length;
    length-=ch.length;
  }

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_allocate: failed to unlock %s",cf->filename);
    return -1;
  }

  return firstchunk;
}



/* free an individual chunk, and return what was its next pointer
 * -- again static so we assume locks have already been claimed
 */
static int chunk_free(chunkfile *cf, chunk_id cid)
{
  chunk_id freepointer, oldnext;
  chunkheader deadchunk;
  int bc=0;

#ifdef DEBUG
  printf("Starting chunk_free((chunkfile *)0x%x,(chunk_id)0x%x)\n" \
        ,(int)cf,(int)cid);
#endif

  /* 1) read current free pointer
   * 2) read existing header for chunk to go
   * 3) deadchunk.length = -1,  deadchunk.next = freepointer
   * 4) set free pointer to cid of chunk we are deleting
   */

  /* read current free pointer */
  fseek(cf->fp, 4, SEEK_SET);        /* jump to head of free list */
  fread(&freepointer, sizeof(chunk_id), 1, cf->fp); /* load value */

  /* read existing header for chunk to go */
  fseek(cf->fp, CID_TO_BYTE(cid,cf->chunksize), SEEK_SET);
  if(!fread(&deadchunk, sizeof(chunkheader), 1, cf->fp)) /* load header */
  {
    fprintf(stderr,"chunk_free: failed to load chunk header 0x%x\n",cid);
    return -1;
  }

  oldnext=deadchunk.next;
  deadchunk.length=-1;
  deadchunk.next=freepointer;

  /* write out new chunk header for chunk we are freeing */
  fseek(cf->fp, CID_TO_BYTE(cid,cf->chunksize), SEEK_SET);
  fwrite(&deadchunk, sizeof(chunkheader), 1, cf->fp);

  bc=cf->chunksize;

  /* clear the data area */
  while(bc)
  {
    fputc(0,cf->fp);
    --bc;
  }

  freepointer=cid;

  /* write new free pointer */
  fseek(cf->fp, 4, SEEK_SET);        /* jump to head of free list */
  fwrite(&freepointer, sizeof(chunk_id), 1, cf->fp);

#ifdef DEBUG
  printf("done, written new free pointer %x\n",freepointer);
#endif

  return oldnext;
}

/* FUTURE : Currently this freeing algorithm results in a inversed
 *          list of chunks being put in the free list, ie a chain of
 *          chunk_ids : 23, 34, 56, 57, 58, 90, 92 will be added to
 *          the front of the free list as 92, 90, 58, 57, 56, 34, 23.
 *          We could read the list of chunks in and free them in the
 *          correct order to avoid this. In the search engine
 *          it is unlikely that this will be a problem, until expiry
 *          etc. is implemented.
 */

void chunk_deallocate(chunkfile *cf, chunk_id cid)
{
#ifdef DEBUG
  printf("Starting chunk_deallocate((chunkfile *)0x%x,(chunk_id)0x%x)\n" \
       ,(int)cf,(int)cid);
  printf("cf->fp=%x\n",(int)cf->fp);
#endif

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_deallocate: failed to lock %s",cf->filename);
    return;
  }

  do {
    cid=chunk_free(cf,cid);
  } while(cid!=-1);

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_deallocate: failed to unlock %s",cf->filename);
    return;
  }

#ifdef DEBUG
  printf("done\n");
#endif
}

/* returns a malloced block and length for the data in a chunk chain.
 * user passed pointers to variables to hold data pointer and length,
 * along with chunkfile handle and start chunk_id for the chain.
 *
 * returns non-zero in case of a problem */

int chunk_retrieve(chunkfile *cf, chunk_id cid, char **data, int *length)
{
  chunkheader ch;
  char *rdata=NULL;
  int rlength=0;

#ifdef DEBUG
  printf("Starting chunk_retrieve((chunkfile *)0x%x,(chunk_id)0x%x"
         ", data, length)\n",(int)cf,(int)cid);
  printf("cf->fp=%x\n",(int)cf->fp);
#endif

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_retrieve: failed to lock %s",cf->filename);
    return -1;
  }

  ch.next=cid;

  while(ch.next != -1)
  {
    int last;

    last=ch.next;

#ifdef DEBUG
    printf("chunk id = 0x%x, byte offset = 0x%x: ",ch.next,CID_TO_BYTE(ch.next,cf->chunksize));
#endif

    /* read header for chunk */
    fseek(cf->fp, CID_TO_BYTE(ch.next,cf->chunksize), SEEK_SET);
    if(!fread(&ch, sizeof(chunkheader), 1, cf->fp)) /* load header */
    {
      fprintf(stderr,"chunk_retrieve: failed to load chunk header 0x%x\n",cid);
      *data=NULL;
      *length=0;
      free(rdata);

      if(unlock_file(cf->fp) == -1)
        fprintf(stderr,"chunk_retrieve: failed to lock %s",cf->filename);

      return -1;
    }
```

```
#ifdef DEBUG
    printf("ch.next = 0x%x, ch.length = %i\n",ch.next,ch.length);
#endif

    if(ch.next==last)
    {
      fprintf(stderr,"chunk_retrieve: ERK! Circular reference in chunk file %s! chunk_id=0x%x\n",cf->filename,last);
      if(unlock_file(cf->fp) == -1)
        fprintf(stderr,"chunk_retrieve: failed to unlock %s",cf->filename);
      return -1;
    }

    /* allocate memory */
    if(rdata)
      rdata=realloc(rdata,rlength+ch.length+1);
    else
      rdata=malloc(ch.length+1);

    if(!rdata)
    {
      fprintf(stderr,"chunk_retrieve: failed m/realloc %i bytes\n",
              rlength+ch.length+1);
      *data=NULL;
      *length=0;
      free(rdata);
      if(unlock_file(cf->fp) == -1)
        fprintf(stderr,"chunk_retrieve: failed to unlock %s",cf->filename);
      return 1;
    }

#ifdef DEBUG
    printf("fread(rdata=0x%x+rlength=%i,ch.length=%i,1,cf->fp=0x%x)\n",(int)rdata,(int)rlength,ch.length,(int)cf-
>fp);
#endif

    /* fread in */
    if(ch.length != (fread(rdata+rlength, 1, ch.length, cf->fp)))
    {
      fprintf(stderr,"chunk_retrieve: bad chunk 0x%x in %s, failed to read %i bytes data.\n",last,cf-
>filename,ch.length);
      *data=0;
      *length=0;
      free(rdata);
      if(unlock_file(cf->fp) == -1)
        fprintf(stderr,"chunk_retrieve: failed to unlock %s",cf->filename);
      return 0;
    }

    rlength+=ch.length;
  }

  /* return values */
  *data=rdata;
  *length=rlength;

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_retrieve: failed to unlock %s",cf->filename);
    return -1;
  }

#ifdef DEBUG
  printf("done\n");
#endif

  return 0;
}

void chunk_alter(chunkfile *cf, chunk_id cid, char *data, int length)
{
}

/* appends data to an existing chain of chunks, returns 0 if successful,
 * non-zero if there was a problem...
 */
int chunk_append(chunkfile *cf, chunk_id cid, char *data, int length)
{
  chunkheader ch;
  chunk_id current;

#ifdef DEBUG
  printf("Starting chunk_append((chunkfile *)0x%x,(chunk_id)0x%x," \
         "(char *)0x%x,(int)%i)\n",(int)cf,(int)cid,(int)data,length);
  printf("cf->fp=%x\n",(int)cf->fp);
#endif

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_append: failed to lock %s",cf->filename);
    return -1;
  }

  ch.length=0;
  ch.next=cid;

  /* find last chunk in chain */

  while(ch.next != -1)
  {
    fseek(cf->fp, CID_TO_BYTE(ch.next,cf->chunksize), SEEK_SET);
    current=ch.next;
    if(!fread(&ch, sizeof(chunkheader), 1, cf->fp)) /* load header */
    {
      fprintf(stderr,"chunk_append: failed to load chunk header 0x%x whilst appending to chunk id
0x%x\n",ch.next,cid);
      if(unlock_file(cf->fp) == -1)
        fprintf(stderr,"chunk_append: failed to unlock %s",cf->filename);
      return 1;
    }
  }

  /* now we need to add our data, creating new chunks as necessary -
   * first fill any remaining space in our end chunk
   *
   * chunksize 10
   *
   * 1234567890.12345
   *            ch.length=5
   *
   * cf-chunksize-ch.length=5;
   *
   * so write first 5 bytes of data to end of this chunk
   */

#ifdef DEBUG
  printf("ch.length == %i, cf->chunksize == %i\n",ch.length,cf->chunksize);
#endif

  if(ch.length < cf->chunksize) /* if this chunk is not full... */
  {
    if(length >= cf->chunksize-ch.length)  /* if we can fill chunk completely */
    {
#ifdef DEBUG
      printf("length (%i) > cf->chunksize-ch.length (%i)\n",length,cf->chunksize-ch.length);
      printf("fpos pre-fseek +%i = 0x%x\n",ch.length,(int)ftell(cf->fp));
#endif
      fseek(cf->fp,ch.length,SEEK_CUR); /* seek to end of current data */
#ifdef DEBUG
      printf("writing at 0x%x\nDATA: [",(int)ftell(cf->fp));
      fwrite(data, cf->chunksize-ch.length,1,stdout);
      printf("]");
#endif
      fwrite(data, cf->chunksize-ch.length,1,cf->fp);
      data+=cf->chunksize-ch.length;
      length-=cf->chunksize-ch.length;
      ch.length+=cf->chunksize-ch.length;
      ch.next=allocate_next_free_chunk(cf); /* we need more chunks, allocate
                                              * one now so ch.next is right */

    } else {
#ifdef DEBUG
      printf("newchunk! length (%i) < cf->chunksize-ch.length (%i)\n",length,cf->chunksize-ch.length);
#endif

      fseek(cf->fp,ch.length,SEEK_CUR); /* seek to end of chunk data */
      fwrite(data, length,1,cf->fp);
      data+=length;
      ch.length+=length;
      length=0;
    }
    fseek(cf->fp, CID_TO_BYTE(current,cf->chunksize), SEEK_SET);
    fwrite(&ch, sizeof(chunkheader), 1, cf->fp); /* update header */
  }

  while(length)
  {
    int newchunk;

    if(ch.next==-1)
      newchunk=allocate_next_free_chunk(cf);
    else
      newchunk=ch.next;

    if(length>(cf->chunksize))
    {
      ch.length=cf->chunksize;
      ch.next=allocate_next_free_chunk(cf);
      //if(ch.next==newchunk)
      //ch.next=newchunk+1;
    } else {
      ch.length=length;
      ch.next=-1;
    }

#ifdef DEBUG
    printf("newchunk = %x newchunk.next = %x\n",newchunk,ch.next);
#endif

    /* fseek, fwrite, etc */
    fseek(cf->fp, CID_TO_BYTE(newchunk,cf->chunksize), SEEK_SET);
#ifdef DEBUG
    printf("writing chunk 0x%x at %i in file\n", newchunk, CID_TO_BYTE(newchunk,cf->chunksize));
    printf("DATA: [");
    fwrite(data, ch.length, 1, stdout);
    printf("]");
#endif
    fwrite(&ch, sizeof(chunkheader), 1, cf->fp);
    fwrite(data, ch.length, 1, cf->fp);
    data=data+ch.length;
    length-=ch.length;
  }

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_append: failed to unlock %s",cf->filename);
    return -1;
  }

  return 0;
}

#ifdef DEBUG
static void chunk_dump_freelist(chunkfile *cf)
{
  chunk_id cid;
  chunkheader ch;

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_dump_freelist: failed to lock %s",cf->filename);
  }

  fseek(cf->fp, 4, SEEK_SET);      /* jump to head of free list */
  fread(&cid, sizeof(chunk_id), 1, cf->fp); /* load value */

  printf("free list ");

  while(cid!=-1)
  {
    printf(" -> 0x%x",cid);
    fseek(cf->fp, CID_TO_BYTE(cid,cf->chunksize), SEEK_SET);
    fread(&ch, sizeof(chunkheader), 1, cf->fp); /* update header */
    cid=ch.next;
  }

  printf("\n");

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"chunk_dump_freelist: failed to lock %s",cf->filename);
  }

}
#else
static void chunk_dump_freelist(chunkfile *cf)
{
  cf=cf;
}
#endif

#ifdef DEBUG
void debug_dump_chunk(chunkfile *cf, chunk_id cid)
{
  char *chunkdata;
  int chunklength;

  if(lock_file(cf->fp) == -1)
  {
    fprintf(stderr,"debug_dump_chunk: failed to lock %s",cf->filename);
  }

  if(chunk_retrieve(cf, cid ,&chunkdata, &chunklength))
  {
    fprintf(stderr,"debug_dump_chunk: Failed to retrieve chunk 0x%x from %s\n",cid,cf->filename);
    return;
  }

  printf("chunk chain starting 0x%x in %s:\n",cid,cf->filename);
  debug_dumper(chunkdata,chunklength,stdout,DUMP_WORD);

  free(chunkdata);

  if(unlock_file(cf->fp) == -1)
  {
    fprintf(stderr,"debug_dump_chunk: failed to unlock %s",cf->filename);
  }

}
#endif


#ifdef TESTCHUNKS

int main(void)
{
   char *mem;
   int length=0,error=0;
   chunkfile *cf;
   chunk_id cid,cid2,cid3;
   char monkey[]="this is a little piece of text which will test to see if we can append on to a existing chunk
chain!";
   char ast[]="that was astf";
   char apc[]="that was apcs, oh yes, very nice.";

   remove("/home/thowat/testchunks");
   cf=chunk_open("/home/thowat/testchunks",1024);
   if(cf==NULL)
     exit(1);
   mem=load_malloc("/home/thowat/project/tests/01frd10.txt", &length, &error);
   printf("length = %i, error = %i\n",length,error);
   if(error)
     exit(1);
   cid=chunk_allocate(cf,mem,length);
   chunk_append(cf,cid,monkey,strlen(monkey));
   chunk_append(cf,cid,monkey,strlen(monkey));
   chunk_append(cf,cid,monkey,strlen(monkey));
   chunk_append(cf,cid,monkey,strlen(monkey));
   chunk_deallocate(cf,cid);
   free(mem);
   chunk_dump_freelist(cf);
   mem=load_malloc("/home/thowat/project/tests/03hgp10.txt", &length, &error);
   printf("length = %i, error = %i\n",length,error);
   if(error)
     exit(1);
   cid=chunk_allocate(cf,mem,length);
   mem=load_malloc("/home/thowat/project/tests/02frd10.txt", &length, &error);
   printf("length = %i, error = %i\n",length,error);
   cid2=chunk_allocate(cf,mem,length);
   //chunk_append(cf,cid2,ast,strlen(ast));
   free(mem);
   chunk_dump_freelist(cf);
   mem=load_malloc("/home/thowat/project/tests/02frd10.txt", &length, &error);
   printf("length = %i, error = %i\n",length,error);
   cid3=chunk_allocate(cf,mem,length);
   free(mem);
   chunk_dump_freelist(cf);
   //mem=load_malloc("$.apcs", &length, &error);
   //cid3=chunk_allocate(cf,mem,length);
   //chunk_append(cf,cid3,apc,strlen(apc));
   //free(mem);
   //chunk_dump_freelist(cf);

   //mem=load_malloc("$.test", &length, &error);
   //cid=chunk_allocate(cf,mem,length);
   //chunk_append(cf,cid,monkey,strlen(monkey));
   //chunk_dump_freelist(cf);
   //free(mem);

   printf("TEST FILE 1!!!!!\n");
   chunk_retrieve(cf,cid,&mem,&length);
   if(mem) printf("%s",mem);
   free(mem);
   printf("TEST FILE 2!!!!!\n");
   chunk_retrieve(cf,cid2,&mem,&length);
   if(mem) printf("%s",mem);
   free(mem);
   printf("TEST FILE 3!!!!!\n");
   chunk_retrieve(cf,cid3,&mem,&length);
   if(mem) printf("%s",mem);
   free(mem);

   chunk_close(cf);

   return 0;
}

#endif
```

F.4 chunks.h

```
/*
 * chunks.h
 * Tony Howat 1999/2000
 *
 * This code administers blocks of data stored in files - these blocks
 * of data are split into "chunks" of uniform length, which are chained
 * together.
 *
 * This give us random-access type performance with variable length
 * records.
 *
 */

#ifndef __CHUNKS_H
#define __CHUNKS_H 1

typedef struct _chunkfile {
  FILE *fp;          /* normal C file pointer for chunk file */
  int chunksize;     /* size of data area per chunk */
  char *filename;    /* filename for reference */
#ifdef DEBUG
  FILE *dp;          /* C file pointer for debug trace file */
#endif
} chunkfile;

typedef int chunk_id;

/* open a chunked file
 */
chunkfile *chunk_open(char *file,int chunksize);

/* close a chunked file
 */
void chunk_close(chunkfile *cf);

/* create a chain of chunks containing data of length pointed to by
 * data, returns -1 in case of failure.
 */
chunk_id chunk_allocate(chunkfile *cf, char *data, int length);

/* chunk_deallocate frees an entire chain of chunks allowing them to
 * be re-used
 */
void chunk_deallocate(chunkfile *cf, chunk_id cid);


/* returns a malloced block and length for the data in a chunk chain.
 * user passed pointers to variables to hold data pointer and length,
 * along with chunkfile handle and start chunk_id for the chain.
 *
 * returns non-zero in case of a problem */

int chunk_retrieve(chunkfile *cf, chunk_id cid, char **data, int *length);
```

```
/* not implemented yet */
void chunk_alter(chunkfile *cf, chunk_id cid, char *data, int length);

/* appends data to an existing chain of chunks, returns 0 if successful,
 * non-zero if there was a problem...
 */
int chunk_append(chunkfile *cf, chunk_id cid, char *data, int length);

#ifdef DEBUG
/* dump a chunk to fp */
void debug_dump_chunk(chunkfile *cf, chunk_id cid);
#endif

#endif
```

F.5 db.c

```
/*
 * db.c
 * Tony Howat 2000
 *
 * The central database code for dealing with words/weightings lists
 * and url database
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"
#include "chunks.h"
#include "cbtree.h"
#include "global.h"
#include "preprocdoc.h"
#include "debug.h"
#include "shavtimer.h"

#define MULWEIGHTS 1

/* document/weighting pairs for docbase */
typedef struct
{
  int weighting;
  chunk_id document;
} DocWeighting;

cbtfile *urlbase;   /* our handle for the url index */
cbtfile *wordbase;  /* our handle for the words/weighting database */
cbtfile *docbase;   /* document database */

/* compiles a string containing the server statistics */
char *db_getstats(int *length)
{
  char *sdata;

  sdata=malloc(2048); /* plenty of space */
  if(!sdata)
  {
    *length=0;
    return NULL;
  }

  sprintf(sdata,"<b>Search engine data server, built"
          " on " __DATE__ " " __TIME__ "</b><br>\n"
          "&copy; Tony Howat 2000<br>\n"
          "Compiled with options :"
#ifdef DEBUG
          " debugging output."
#endif
#ifdef VIRTUALTIME
          " virtual timers."
#endif
#ifdef STEMMING
          " stemming."
#endif
#ifdef PROFILING
          " profiling."
#endif
#ifdef FILESTATS
          " filestats."
#endif
          "<p>\n"
          "<table border=\"1\" width=\"100%\">\n"
          "<tr><td>Abstract length<td>%i\n"
          "<tr><td>Title length<td>%i\n"
          "<tr><td>Max terms per document<td>%i\n"
          "<tr><td>URL chunk length<td>%i\n"
          "<tr><td>Word file chunk length<td>%i\n"
          "<tr><td>Term length<td>%i\n"
          "<tr><td>Number of documents indexed<td>%i\n"
          "<tr><td>Number of unique terms indexed<td>%i\n"
          "<tr><td>Retrieves serviced since server start<td>%i\n"
          "<tr><td>Average time per retrieve<td>%ims (last took %ims)\n"
          "<tr><td>Searches serviced since server start<td>%i\n"
          "<tr><td>Average time per search<td>%ims (last took %ims)\n"
          "<tr><td>Submits serviced since server start<td>%i\n"
          "<tr><td>Average time per submit<td>%ims (last took %ims)\n"
          "</table>\n<p>\n",
          ABSLEN,TITLELEN,MAXTERMS,URLCHUNKLENGTH,WORDFILECHUNKLENGTH,
          TERMSTRINGLENGTH,
          cbtree_countentries(urlbase),
          cbtree_countentries(wordbase),
          shavtimer_getoperations(shavt_retrieve),
          shavtimer_getaverage(shavt_retrieve),
           shavtimer_getlast(shavt_retrieve),
          shavtimer_getoperations(shavt_search),
          shavtimer_getaverage(shavt_search),
          shavtimer_getlast(shavt_retrieve),
          shavtimer_getoperations(shavt_submit),
          shavtimer_getaverage(shavt_submit),
          shavtimer_getlast(shavt_retrieve));

  *length=strlen(sdata);
  return sdata;
}

#ifdef FILESTATS
char *data;

/* get length of a particular file */
static int flength(char *file)
{
  FILE *f;
  int len;

  f=fopen(file,"r");
  if(f==NULL)
    return 0;
  fseek(f,0,SEEK_END);
  len=ftell(f);

  return len;
}

/* dump file size statistics to STATSFILE - purely for metrics */
void db_dump_filestats()
{
  FILE *f;
  char fn[128];
  char *urlfile, *wordfile, *docfile;

  urlfile=malloc(strlen(DBROOTFILE)+strlen(".urls")+strlen(".index")+1);
  wordfile=malloc(strlen(DBROOTFILE)+strlen(".words")+strlen(".index")+1);
  docfile=malloc(strlen(DBROOTFILE)+strlen(".docs")+strlen(".index")+1);

  if((!urlfile) | (!wordfile) | (!docfile))
  {
    fprintf(stderr,"db_dump_filestats: failed to malloc() for filenames!\n");
    return;
  }

  sprintf(urlfile,"%s.urls",DBROOTFILE);
  sprintf(wordfile,"%s.words",DBROOTFILE);
  sprintf(docfile,"%s.docs",DBROOTFILE);

  if(flength(STATSFILE))
  {
    f=fopen(STATSFILE,"a+");
    if(f)
      fseek(f,0,SEEK_END);
  } else {
    f=fopen(STATSFILE,"w");
    if(f)
      fprintf(f,"Documents,Words,urlbase,wordbase,docbase,"
                "urlfile.index,wordfile.index,docfile.index,"
                "AverageSubmitTime\n");
  }

  if(!f)
  {
    log_event("db_dump_filestats: Failed to write stats");
    return;
  }

  fprintf(f,"%i,%i,%i,%i,%i,",cbtree_countentries(urlbase),
          cbtree_countentries(wordbase),flength(urlfile),
          flength(wordfile),flength(docfile));
  strcat(urlfile,".index");
  strcat(docfile,".index");
  strcat(wordfile,".index");
  fprintf(f,"%i,%i,%i,%i\n",flength(urlfile),flength(wordfile),
                        flength(docfile),shavtimer_getaverage(shavt_submit));
  fclose(f);
}
#endif


/* db start - returns non-zero if there is a fault */
int db_start(char *file)
{
  char *urlfile,*wordfile,*docfile;

  urlfile=malloc(strlen(file)+strlen(".urls")+1);
  wordfile=malloc(strlen(file)+strlen(".words")+1);
  docfile=malloc(strlen(file)+strlen(".docs")+1);

  if((!urlfile) | (!wordfile) | (!docfile))
  {
    fprintf(stderr,"db_start: failed to malloc() for filenames!\n");
    return -1;
  }

  sprintf(urlfile,"%s.urls",file);
  sprintf(wordfile,"%s.words",file);
  sprintf(docfile,"%s.docs",file);

  if((urlbase=cbtree_open(urlfile,URLCHUNKLENGTH))==NULL)
  {
    fprintf(stderr,"db_start: failed to open %s!\n",urlfile);
    free(urlfile);
    free(docfile);
    free(wordfile);
    return -1;
  }

  docbase=cbtree_open(docfile,sizeof(DocumentRecord)+
                      (sizeof(chunk_id)*(MAXTERMS+1)));
  if(docbase==NULL)
  {
    fprintf(stderr,"db_start: failed to open %s!\n",docfile);
    free(urlfile);
    free(docfile);
    free(wordfile);
    return -1;
  }

  free(urlfile);
  free(docfile);

  if((wordbase=cbtree_open(wordfile,WORDFILECHUNKLENGTH))==NULL)
  {
    fprintf(stderr,"db_start: failed to open %s!\n",wordfile);
    free(wordfile);
    return -1;
  }

  free(wordfile);
  return 0;
}

/* retrieves a DocumentRecord structure from the database for
 * the document with the base-form url hash of the same number.
 * Appended to this structure is the url of the document.
 *
 * returns non-zero in case of failure
 */
int db_get_docrecord(int hash, chunk_id c, DocumentRecord **dro, int *length)
{
  DocumentRecord *drt=NULL, *drn=NULL;
  int len=0, urll=0;
  char *urldata=NULL;
  char *url=NULL;

  /* start timing */
  shavtimer_starttimer(shavt_retrieve);

  if(!c)
  {
    if(cbtree_retrieve(docbase,hash,(char **)&drt,&len,-1)==-1)
    {
      fprintf(stderr,"db_get_docrecord: cbtree_retrieve of "
                     "DocRecord with hash 0x%x failed\n",hash);
      shavtimer_droptimer(shavt_retrieve);
      return -1;
    }
  } else {
    if(chunk_retrieve(cbtree_getchunkfilehandle(docbase),c,(char **)&drt,&len))
    {
      fprintf(stderr,"db_get_docrecord: chunk_retrieve of "
                     "DocRecord with chunk_id 0x%x failed\n",c);
      shavtimer_droptimer(shavt_retrieve);
      return -1;
    }
  }

  /* now drt contains the DocumentRecord followed by a list of
   * chunk_ids of each term that is indexed in the file. These
   * terms are no use to the external program, it needs the url
   * instead, so we look up the URL string, allocate memory for
   * it and the DocumentRecord, copy the data in, and free both
   * the old DocumentRecord and the URL which we looked up.
   */

  /* 1. look up the url
   */
  if(cbtree_retrieve(urlbase,drt->urlhash,&urldata,&urll,-1)==-1)
  {
    fprintf(stderr,"db_get_docrecord: cbtree_retrieve of "
                   "url with hash 0x%x failed\n",drt->urlhash);
    free(drt);
    shavtimer_droptimer(shavt_retrieve);
    return -1;
  }

  /* 2. allocate memory for url and DocumentRecord, and construct it */
  len=sizeof(DocumentRecord)+strlen(urldata)+1;

  drn=(DocumentRecord *)malloc(len);
  if(!drn)
  {
    fprintf(stderr,"db_get_docrecord: Failed to get docrecord, couldn't "
            "malloc %i bytes for DocumentRecord\n",len);
    free(drt);
    free(urldata);
    shavtimer_droptimer(shavt_retrieve);
    return -1;
  }

  memcpy(drn,drt,sizeof(DocumentRecord));
  url=(char *)drn+(sizeof(DocumentRecord)); /* find where url will go */
  strcpy(url,urldata);

  /* 3. free old stuff */
  free(drt);
  free(urldata);

  shavtimer_stoptimer(shavt_retrieve);

  /* 4. output values + return */
  *dro=drn;
  *length=len;
  return 0; /* OK */
}

/* submits a document to the database, returns non-zero in case of fault */
int db_submit_document(PreProcDoc *ppdn)
{
  DocumentRecord *dr;

  char *url;
  int urlindex;         /* index pointer for URL */
  chunk_id urlchunk;    /* head chunk for URL */
  int docindex;         /* index pointer for DocumentRecord */
  chunk_id docchunk;    /* head chunk for DocumentRecord */
  WordEntry *we;        /* head of WordEntries */
  chunk_id *termchunks; /* list of terms in a document (chunk_ids) */
  int termchunksindex=0;
  int i=0;

  shavtimer_starttimer(shavt_submit);

  /* to submit a document we create a DocumentRecord object
   * followed by enough space for ppdn->indexedwords chunk_ids
   * we fill in the record, and fill in the extra space with
   * every chunk_id we find/allocate for a word. We create
   * the URL record.
   */

  /* allocate memory for document record + word list */
  dr=malloc((sizeof(chunk_id)*ppdn->indexedwords)+sizeof(DocumentRecord));

  if(!dr)
  {
    fprintf(stderr,"db_submit_document: Failed to submit document, couldn't "
           "malloc %i bytes for DocumentRecord\n",
           (sizeof(chunk_id)*ppdn->indexedwords)+sizeof(DocumentRecord));
    shavtimer_droptimer(shavt_submit);
    return -1;
  }

  /* we add one to the pointer as adding one to a specifically typed
   * pointer (in this case WordEntry *) will add sizeof(type) - not
   * 1
   */
  we=(WordEntry *)(ppdn+1);

  /* get pointer to url from afterPreProcDoc structure */
  url=(char *)we+(sizeof(WordEntry)*ppdn->indexedwords);

  strcpy(dr->title,ppdn->title);
  strcpy(dr->abstract,ppdn->abstract);
  dr->time=ppdn->time;
  dr->urlhash=expohash(url,strlen(url));
  dr->length=ppdn->doclength;

  /* look for this url to make sure it hasn't been submitted already */
  urlindex=cbtree_locate(urlbase,dr->urlhash);

  if(urlindex!=-1)
  {
    /* This would eventually look into replacing the data if necessary */
    fprintf(stderr,"db_submit_document: Failed to submit document, URL \"%s\""
           "appears to have already been submitted\n",url);
    free(dr);
    shavtimer_droptimer(shavt_submit);
    return -1;
  } else {
    /* create a new url entry */
    urlchunk=cbtree_allocate(urlbase,dr->urlhash,url,strlen(url));
#ifdef DEBUG
    printf("url %s (0x%x) given chunk 0x%x\n",url,dr->urlhash,urlchunk);
    debug_dump_file(urlbase->filen,DUMP_WORD);
    printf("-oOo-\n");
#endif
    if(urlchunk==-1)
    {
      fprintf(stderr,"db_submit_document: Failed to allocate chunk for url\n");
      shavtimer_droptimer(shavt_submit);
      return -1;
    }
  }

  dr->sourceurl=urlchunk;

  /* Now write the head of the DocumentRecord structure and get a
   * chunk_id for it
   */

  /* look for this url to make sure it hasn't been submitted already */
  docindex=cbtree_locate(docbase,dr->urlhash);

  if(docindex!=-1)
  {
    /* This would eventually look into replacing the data if necessary */
    fprintf(stderr,"db_submit_document: Failed to submit document, URL \"%s\""
           "appears to have already been submitted (DocumentRecord)\n",url);
    free(dr);
    shavtimer_droptimer(shavt_submit);
    return -1;
  } else {
    /* create a new DocumentRecord entry */
    docchunk=cbtree_allocate(docbase,dr->urlhash,(char *)dr,
                         sizeof(DocumentRecord));
    if(docchunk==-1)
    {
      fprintf(stderr,"db_submit_document: Failed to allocate chunk for"
                    " DocumentRecord\n");
      shavtimer_droptimer(shavt_submit);
      return -1;
    }
  }

  /* Now we have a DocumentRecord header, which is complete. Run through
   * submitted terms and weightings. for each one :
   *  1) Add to wordbase
   *  2) Store chunk_id after the DocumentRecord
   * chunk_ids for after the DocumentRecord are stored in memory and
   * written after all the words/weightings have been added to save on
   * disc operations.
   */

  /* get to the space reserved for chunk_ids of terms
   * in a document
   */
  termchunks=(chunk_id *)(dr+1);

  /* main loop */
  while(i!=ppdn->indexedwords)
  {
    DocWeighting dw;
    int termhash;
    int termindex=0;
    chunk_id termchunk=-1;

#ifdef DEBUG
    printf("word = %s, occurrences = %i, weighting = %i\n",
           we[i].word, we[i].occurrences, we[i].weighting);
#endif

    /* search for existing term in index */
    termhash=expohash(we[i].word,strlen(we[i].word));
    termindex=cbtree_locate(wordbase,termhash);

    if(termindex==-1)
    {
      /* create a new term entry */
      termchunk=cbtree_allocate(wordbase,termhash,
                         we[i].word,TERMSTRINGLENGTH);
      if(termchunk==-1)
      {
        fprintf(stderr, "db_submit_document: Failed to allocate chunk for "
                "term %s in document %s\n",we[i].word,url);
        free(dr);
        shavtimer_droptimer(shavt_submit);
        return -1;
      }
#ifdef DEBUG
      printf("creating new entry...\n");
      debug_dump_chunk(cbtree_getchunkfilehandle(wordbase),termchunk);
#endif
    } else {
      termchunk=cbtree_getchunk(wordbase,termindex);
#ifdef DEBUG
      printf("existing...\n");
      debug_dump_chunk(cbtree_getchunkfilehandle(wordbase),termchunk);
#endif
    }

    dw.weighting=we[i].occurrences*(we[i].weighting+1);
    dw.document=docchunk;

    /* add this document and weighting to list for this term */
    if(cbtree_append(wordbase,termhash,(char *)&dw,sizeof(DocWeighting),-1))
    {
      fprintf(stderr, "db_submit_document: Failed to add docchunk 0x%x for"
              " term %s in document %s\n",docchunk,we[i].word,url);
      free(dr);
      shavtimer_droptimer(shavt_submit);
      return -1;
    }

#ifdef DEBUG
    printf("appended...\n");
    debug_dump_chunk(cbtree_getchunkfilehandle(wordbase),termchunk);
#endif

    /* finally add the chunk_id of this term to our DocumentRecord */
    termchunks[termchunksindex]=termchunk;
#ifdef DEBUG
    printf("termchunks[%i]=%i\n",termchunksindex,termchunks[termchunksindex]);
#endif
    ++termchunksindex;

    ++i;
  }

#ifdef DEBUG
  printf("Pre-appending chunk_ids to DocumentRecord:\n");
  debug_dump_chunk(cbtree_getchunkfilehandle(docbase),cbtree_locate(docbase,dr->urlhash));
#endif

  /* now append the chunk_ids to our DocumentRecord */
  if(cbtree_append(docbase,dr->urlhash,(char *)termchunks,
     termchunksindex*sizeof(chunk_id),docindex))
  {
    fprintf(stderr,"db_submit_document: Failed to append chunk_ids to for "
            "document %s (chunk_id=0x%x)\n",url,docchunk);
    free(dr);
    shavtimer_droptimer(shavt_submit);
    return -1;
  }

#ifdef DEBUG
  printf("Post-appending...");
  debug_dump_chunk(cbtree_getchunkfilehandle(docbase),cbtree_locate(docbase,dr->urlhash));
#endif

#ifdef TERMDEBUG
  /********************
   * now a bit of debugging code to dump every term's references
   ********************/

  {
    i=0;

    while(i!=ppdn->indexedwords)
    {
      char *chunkdata;
      int chunklength;
      DocWeighting *dwl;

      /* 1 load up the chunk for this word
       * 2 dump it out to stdout
       */

      if(chunk_retrieve(cbtree_getchunkfilehandle(wordbase),termchunks[i],
                        &chunkdata, &chunklength))
      {
        fprintf(stderr,"db_submit_document DEBUG: Failed to retrieve chunk 0x%x from wordbase\n",termchunks[i]);
        return -1;
      }

      debug_dumper(chunkdata,chunklength,stdout,DUMP_WORD);

      printf("%s referenced by... \n",chunkdata);

      dwl=(DocWeighting *)((char *)chunkdata+TERMSTRINGLENGTH);

      printf("while(dwl(0x%x)!=(chunkdata(0x%x)+chunklength(%i))\n",
             dwl,chunkdata,chunklength);

      while((char *)dwl!=((char *)chunkdata)+chunklength)
      {
        int urll;
        char *urldata;

        printf("Line %i dwl == 0x%x\n",__LINE__,dwl);

        printf("(0x%x dwl.document=0x%x term=0x%x)",dwl,dwl->document,chunkdata+chunklength);

        if(chunk_retrieve(cbtree_getchunkfilehandle(urlbase),dwl->document,
           &urldata,&urll))
        {
          fprintf(stderr,"db_submit_document DEBUG: Failed to retrieve chunk 0x%x from urlbase\n",dwl->document);
        } else {
          urldata[urll]='\0';
          printf("URL[%s] (0x%x) -> \n",urldata,dwl->document);
          free(urldata);
        }
        printf("dwl now == 0x%x\n",dwl);
        dwl++;
        printf("dwl now == 0x%x\n",dwl);
      }

      free(chunkdata);

      ++i;
    }
  }

#endif

  free(dr);

  shavtimer_stoptimer(shavt_submit);
#ifdef FILESTATS
  /* every FILESTATSTEP submissions dump stats */
  if(!(cbtree_countentries(urlbase) % FILESTATSTEP))
  {
    db_dump_filestats();
    /* the following line causes average timings for every
     * FILESTATSTEP submissions - this means averages are
     * not biased by ultra-quick ops on a small database
     */
    shavtimer_reset(shavt_submit);
  }
#endif

  /* and that's about it */
  return 0;
}

/* this function is used by the qsort function to determine which of
 * two objects should be treated as greater
 */
static int ResultSortCompare(const void *a, const void *b)
{
  int *aa, *bb;

  aa=(int *)a;
  bb=(int *)b;
  return(bb[1]-aa[1]);
}

/* perform a query on the database
 * terms is a count of the number of terms, which are presented as a
 * sequence of fixed length (TERMSTRINGLENGTH) strings. The results
 * are returned in a malloced block of integers containing a document
 * indentifier and relevance rating for each submitted term. The
 * resultsl parameter's pointee is filled with the number of bytes
 * in returned block
 * the function returns non-zero in the case of an error.
 */
int db_lookup(int nterms, char *queryblock, int **results, int *resultsl)
{
  /* This is quite a complex procedure. We do a query on each
   * individual term, to identify the documents containing it.
   * We keep the results for each term in an array. Once all the
   * terms have been queried we process all the results:
   *
   *--- for full weighted queries we would do the following, but we
   *--- do not have weighted queries as yet, so the simple solution
   *--- is to total the weightings for each document
   *
   * We create an array of [nterms+2][documents] which is our vector.
   * with space for each DocumentID and a final score to be filled in.
   * For each document in the results fill in a "line" in our vector :
   *
   * DocumentID  term1weight  term2weight  term3weight  .. termNweight
   *
   * We look up the weighting for each term for every document in
   * our buffered results, if it doesn't appear we set it to -1.
   *
   * We end up with a vector looking something like:
   *
   * 0x356abc12      56          -1           -1              -1
   * 0x56a529ab      12          -1           24              9
   * 0x123ab1c2      34           2           12              8
   *
   * We can now free the original results, and start playing with
   * the vector as per section 3.4.2 of Report 1, calculating its
   * distance from our query vector.
   */

  /* Well, that's over with, let's get on with the simple unweighted
   * query code
   */

  DocWeighting **dw;  /* these are the docweighting list pointers for
                       * each term */
  char **cps;         /* these are the original pointers for the word
                       * /weighting chunks so we can free them */
  int *dwlengths;
  int *doctotals;
  int curterm=0;
  int *tresults=0;
  int tresultsl=0;

  shavtimer_starttimer(shavt_search);

  /* 1. create an array of DocWeighting pointers, one for each
   *    query term, an array of lengths, and an array for holding
   *    the original chunk data pointers
   */

  dw=malloc(sizeof(DocWeighting *)*nterms);
  if(dw==NULL)
  {
    fprintf(stderr,"db_lookup: No memory for %i DocWeightings (%i bytes)\n",
            nterms,sizeof(DocWeighting *)*nterms);
    shavtimer_droptimer(shavt_search);
    return -1;
  }

  cps=malloc(sizeof(char *)*nterms);
  if(cps==NULL)
  {
    fprintf(stderr,"db_lookup: No memory for %i char *'s (%i bytes)\n",
            nterms,sizeof(DocWeighting *)*nterms);
    free(dw);
    shavtimer_droptimer(shavt_search);
    return -1;
  }

  dwlengths=malloc(sizeof(int)*nterms);
  if(dwlengths==NULL)
  {
    fprintf(stderr,"db_lookup: No memory for DocWeighting lengths "
            "(%i bytes)\n",sizeof(int)*nterms);
    free(dw);
    free(cps);
    shavtimer_droptimer(shavt_search);
    return -1;
  }

  /* 2. perform a lookup for every term */
  while(curterm!=nterms)
  {
    char *term;
    int hash;

    /* find offset to term string */
    term=queryblock+(curterm*TERMSTRINGLENGTH);

    /* find hash of term */
    hash=expohash(term,strlen(term));
    if(cbtree_retrieve(wordbase,hash,&(cps[curterm]),&(dwlengths[curterm]),-1)==-1)
    {
      fprintf(stderr,"db_lookup DEBUG: No entry for %s\n",term);
      dw[curterm]=NULL;
      dwlengths[curterm]=0;
      cps[curterm]=NULL;
    } else {
      char *tmp;

#ifdef DEBUG
      printf("dw[%i] :\n",curterm);
      debug_dumper(cps[curterm],dwlengths[curterm],stdout,DUMP_WORD);
#endif

      /* drop the string part of the chunk */
      dwlengths[curterm]-=TERMSTRINGLENGTH;
      tmp=cps[curterm];
      tmp=tmp+TERMSTRINGLENGTH;
      dw[curterm]=(DocWeighting *)tmp;
    }

    ++curterm;
  }

  /* now we need to, for every document mentioned in every
   * individual query, create an entry in our results total
   * table - whilst we are doing this we also total the
   * weightings for each document.
   */

  curterm=0;
  while(curterm!=nterms)
  {
    int dwc=0, dwp=0;

    dwc=dwlengths[curterm]/sizeof(DocWeighting);

    while(dwp!=dwc)
    {
      int rc=0; /* results index counter */

      while(rc!=tresultsl)
      {
        if(tresults[rc*2]==dw[curterm][dwp].document)
    {
#ifndef MULWEIGHTS
          /* add on the weighting to existing entry */
          tresults[(rc*2)+1]+=dw[curterm][dwp].weighting;
#else
          /* multiply the weighting with existing entry */
          tresults[(rc*2)+1]*=dw[curterm][dwp].weighting;
#endif
          break;
        }
        ++rc;
      }

      /* no match was found so create a new entry */
      if(rc==tresultsl)
      {
        ++tresultsl;
        tresults=(int *)realloc(tresults,(sizeof(int)*2)*tresultsl);
        if(tresults==NULL)
        {
          fprintf(stderr,"db_lookup: failed to realloc tresults"
                  " to %i bytes!\n", (sizeof(int)*2)*tresultsl);
          shavtimer_droptimer(shavt_search);
          return -1;
        }
        tresults[(tresultsl-1)*2]=dw[curterm][dwp].document;
        tresults[((tresultsl-1)*2)+1]=dw[curterm][dwp].weighting;
      }

      ++dwp;
    }

    ++curterm;
  }

#ifdef DEBUG
  printf("results :\n");
  if(results)
    debug_dumper(tresults,tresultsl,stdout,DUMP_WORD);
  else
    printf("(none)\n");
#endif

  /* finally sort the results into best-match-first */
  qsort((void *)tresults,tresultsl,sizeof(int)*2,ResultSortCompare);

  /* free the temporary areas */
  curterm=0;

  while(curterm!=nterms)
  {
    if(cps[curterm]!=0)
      free(cps[curterm]);
    ++curterm;
  }

  free(dw);
  free(cps);
  free(dwlengths);

  shavtimer_stoptimer(shavt_search);

  /* pass out to the pointed return variables */
  *results=tresults;
  *resultsl=(tresultsl*2)*sizeof(int);
  return 0;
}

/* db finish */
void db_finish(void)
{
  cbtree_close(urlbase);
  cbtree_close(wordbase);
  cbtree_close(docbase);
}
```

F.6 db.h

```
/*
 * db.h
 * Tony Howat 2000
 *
 * The central database code for dealing with words/weightings lists
 * and url database
 */

#include "preprocdoc.h"
#include "chunks.h"

/* db start - returns non-zero if there is a fault */
int db_start(char *file);

/* submits a document to the database, returns non-zero in case of fault */
int db_submit_document(PreProcDoc *ppdn);

/* retrieves a DocumentRecord structure from the database for
 * the document with the base-form url hash of the same number.
 * or if a chunk_id is supplied it will retrieve directly - note
 * that this chunk_id refers to the DocRecords file.
 * Appended to this structure is the url of the document.
 *
 * returns non-zero in case of failure
 */
int db_get_docrecord(int hash, chunk_id c, DocumentRecord **dro, int *length);

/* perform a query on the database
 * terms is a count of the number of terms, which are presented as a
 * sequence of fixed length (TERMSTRINGLENGTH) strings. The results
 * are returned in a malloced block of integers containing a document
 * indentifier and relevance rating for each submitted term.
 * the function returns non-zero in the case of an error.
 */
int db_lookup(int nterms, char *queryblock, int **results, int *resultsl);

/* db finish */
void db_finish(void);

/* compiles a string containing the server statistics */
char *db_getstats(int *length);
```

F.7 debug.c

```
/*
 * debug.c
 * Tony Howat, 1999/2000
 *
 * general debugging routines which are shared between programs
 *
 */

#include <stdio.h>
#include <string.h>
#include <time.h>
#include "preprocdoc.h"
#include "debug.h"
#include "chunks.h"

#ifdef DEBUG

/* we shouldn't need this. temporary measure */
int chunk_retrieve(chunkfile *cf, chunk_id cid, char **data, int *length);

/****** This is code for producing hex dumps of blocks of memory
 ****** application specific debug stuff follows this....
 */

#define TRUE 1
#define STARTPOINT 10
#define NODISSTART 59

#define NODISSTART_C 23
#define notprintable(x) ((x<32) || ((x>126) && (x<129)) || \
                         ((x>130) && (x<133)) || ((x>134) && (x<140)))
#define npchar '.'

char hextable[]="0123456789ABCDEF";

/* This code isn't commented too much as it is self explanatory
 * assuming knowledge of shifts etc. in C.
 */

static void d4w_byte(char *data, int wordpos,int length,FILE *out)
{
  int pos=0;
  char output[80]="                                             "\
                  "                                       \n";

  output[7]=hextable[wordpos & 0xf];
  output[6]=hextable[(wordpos & 0xf0)>>4];
  output[5]=hextable[(wordpos & 0xf00)>>8];
  output[4]=hextable[(wordpos & 0xf000)>>12];
  output[3]=hextable[(wordpos & 0xf0000)>>16];
  output[2]=hextable[(wordpos & 0xf00000)>>20];
  output[1]=hextable[(wordpos & 0xf000000)>>24];
  output[0]=hextable[(wordpos & 0xf0000000)>>28];

  for(pos=0; pos<16; pos++) {
    if(pos<=length) {
      output[(pos*3)+1+STARTPOINT]=hextable[data[pos] & 0xf];
      output[(pos*3)+STARTPOINT]=hextable[(data[pos] & 0xf0)>>4];
      output[NODISSTART+pos]=data[pos];
      if(notprintable(output[NODISSTART+pos]))
        output[NODISSTART+pos]=npchar;
    } else {
      output[STARTPOINT+(pos*3)+1]='-';
      output[STARTPOINT+(pos*3)]='-';
    }
  }

  fputs(output,out);
  return;
}

static void d4w_word(char *data, int wordpos,int length,FILE *out,int bigend)
{
  int pos=0;
  int tab[]={0,9,18,27,36,45,54,63,72}; /* yuk! */
  int tabpos=0;
  char output[80]="                                             "\
                  "                                          \n";
  output[7]=hextable[wordpos & 0xf];
  output[6]=hextable[(wordpos & 0xf0)>>4];
  output[5]=hextable[(wordpos & 0xf00)>>8];
  output[4]=hextable[(wordpos & 0xf000)>>12];
  output[3]=hextable[(wordpos & 0xf0000)>>16];
  output[2]=hextable[(wordpos & 0xf00000)>>20];
  output[1]=hextable[(wordpos & 0xf000000)>>24];
  output[0]=hextable[(wordpos & 0xf0000000)>>28];

  for(pos=0; pos<20; pos++)
  {
    if(pos%4 == 0)
    {
      if(pos>=length)
      {
        int n=7;
        while(n>=0)
        {
          output[STARTPOINT+tab[tabpos]+n]='-';
          --n;
        }
      } else {
        if(!bigend)
        {
          output[STARTPOINT+tab[tabpos]+7]=hextable[data[(pos)] & 0xf];
          output[STARTPOINT+tab[tabpos]+6]=hextable[(data[(pos)] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+5]=hextable[(data[(pos)+1] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]+4]=hextable[(data[(pos)+1] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+3]=hextable[(data[(pos)+2] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]+2]=hextable[(data[(pos)+2] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+1]=hextable[(data[(pos)+3] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]]  =hextable[(data[(pos)+3] & 0xf0)>>4];
        } else {
          output[STARTPOINT+tab[tabpos]]=hextable[data[(pos)] & 0xf];
          output[STARTPOINT+tab[tabpos]+1]=hextable[(data[(pos)] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+2]=hextable[(data[(pos)+1] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]+3]=hextable[(data[(pos)+1] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+4]=hextable[(data[(pos)+2] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]+5]=hextable[(data[(pos)+2] & 0xf0)>>4];
          output[STARTPOINT+tab[tabpos]+6]=hextable[(data[(pos)+3] & 0xf)>>0];
          output[STARTPOINT+tab[tabpos]+7]  =hextable[(data[(pos)+3] & 0xf0)>>4];
        }
      }
      ++tabpos;
    }

    if(pos>length)
    {
      output[STARTPOINT+46+pos]=' ';
    } else {
      if(notprintable(data[pos]))
        output[STARTPOINT+46+pos]=npchar;
      else
        output[STARTPOINT+46+pos]=data[pos];
    }
  }

  fputs(output,out);
  return;
}

/* debug_dumper dumps a block of memory of <size> to a file handle
 * (possibly stdout or stderr) as a hex dump. It does this in a
 * number of formats : DUMP_BYTE     = byte by byte hex
 *                     DUMP_WORD     = as little endian 32 bit words
 *                     DUMP_WORDBIG  = as big endian 32 bit words
 */

void debug_dumper(void *startptr, int size, FILE *outf, int type)
{
  char data[22];
  char *ptr;
  int ng,len;
  int done;
  int unitsize=16;
  int offset;

  offset=(int)startptr;

  ptr=(char *)startptr;

  if(type==DUMP_WORDBIG || type==DUMP_WORD)
    unitsize=20;

  done=0;

  while(ptr < ((char *)startptr+size)) {
    ng=0;len=0;
    while(ng<unitsize) {
      data[ng]=*ptr;
      ++ptr;
      ++ng;
      if( ptr < (char *)startptr+size ) {
        ++len;
      }
    }
    data[ng]='\0';
    if(len)
    {
      switch(type) {
         case DUMP_BYTE:
            d4w_byte(data,offset,len,outf);
            break;
         case DUMP_WORD:
            d4w_word(data,offset,len,outf,0);
            break;
         case DUMP_WORDBIG:
            d4w_word(data,offset,len,outf,1);
            break;
      }
    }
    offset=offset+unitsize;
    done=done+unitsize;
  }
}

/* dump a file's contents to stdout */
void debug_dump_file(char *file,int type)
{
  char *data;
  int length,error;

  data=(char *)load_malloc(file,&length,&error);

  if(!error)
    debug_dumper(data,length,stdout,type);
  else {
    fprintf(stderr,"debug_dump_file: Failed to load %s\n",file);
    return;
  }

  free(data);
}


/****** APPLICATION SPECIFIC DEBUG CODE STARTS HERE ******/

void debug_dump_preprocdoc(PreProcDoc *ppdn)
{
  /* This section of code will extract a PreProcDoc block to the
   * terminal
   */

  char *t;
  int i=0;
  WordEntry *we;
  char *url;

  t=ctime(&ppdn->time);
  t[strlen(t)-1]='\0'; /* get rid of \n */

  printf("PreProcDoc at 0x%x\n",ppdn);
  printf("title = %s\nabstract = %s\ntime = %s\ndoclength = %i\n"
         "wordcount = %i\nindexedwords = %i\n",ppdn->title,
         ppdn->abstract,t,ppdn->doclength,ppdn->wordcount,
         ppdn->indexedwords);

  //we=(WordEntry *)ppdn+sizeof(PreProcDoc);

  /* we add one to the pointer as adding one to a specifically typed
   * pointer (in this case WordEntry *) will add sizeof(type) - not
   * 1
   */
  we=(WordEntry *)(ppdn+1);

  printf("WordEntry list start at 0x%x (sizeof(PreProcDoc)==%i)\n",(int)we,sizeof(PreProcDoc));

  while(i!=ppdn->indexedwords)
  {
    printf("word = %s, occurrences = %i, weighting = %i\n",
           we[i].word, we[i].occurrences, we[i].weighting);
    ++i;
  }
  url=(char *)we+(sizeof(WordEntry)*ppdn->indexedwords);
  printf("url (start @ 0x%x) = %s - end\n",(int)url,url);
}

#endif
```

F.8 debug.h

```
/*
 * debug.h
 * Tony Howat, 1999/2000
 *
 * general debugging routines which are shared between programs
 *
 */

#include "global.h"

#ifdef DEBUG

#include "preprocdoc.h"
#include "chunks.h"

#define DUMP_BYTE    0
#define DUMP_WORD    1
#define DUMP_WORDBIG 2

/* debug_dumper dumps a block of memory of <size> to a file handle
 * (possibly stdout or stderr) as a hex dump. It does this in a
 * number of formats : DUMP_BYTE     = byte by byte hex
 *                     DUMP_WORD     = as little endian 32 bit words
 *                     DUMP_WORDBIG  = as big endian 32 bit words
 */

void debug_dumper(void *startptr, int size, FILE *out, int type);

/* dump a file's contents to stdout */
void debug_dump_file(char *file,int type);

/* extrudes a PreProcDoc to stdout */
void debug_dump_preprocdoc(PreProcDoc *ppdn);

#endif
```

F.9 entities.c

```
/*
 * entities.c
 * Tony Howat, 1999/2000
 *
 * This is a simple piece of code to use a standard HTML 2.0 entity list
 * to translate blocks of text with HTML entities with them.
 *
 */

#include <stdio.h>
#include <string.h>
#include "util.h"

/*

   !-- Portions (c) International Organization for Standardization 1986
        Permission to copy in any form is granted for use with
        conforming SGML systems and applications as defined in
        ISO 8879, provided this notice is included in all copies.
   -->
   <!-- Character entity set. Typical invocation:
        <!ENTITY % HTMLlat1 PUBLIC
          "-//W3C//ENTITIES Latin 1//EN//HTML">
        %HTMLlat1;
   -->

*/

typedef struct _EntityEntry
{
  char *name;   /* entity name */
  int value;    /* char value */
} EntityEntry;

static EntityEntry EntityNames[] =
{
  {"amp",     38},  /* ampersand */
  {"nbsp",    160}, /* no-break space = non-breaking space, U+00A0 ISOnum */
  {"iexcl",   161}, /* inverted exclamation mark, U+00A1 ISOnum */
  {"cent",    162}, /* cent sign, U+00A2 ISOnum */
  {"pound",   163}, /* pound sign, U+00A3 ISOnum */
  {"curren",  164}, /* currency sign, U+00A4 ISOnum */
  {"yen",     165}, /* yen sign = yuan sign, U+00A5 ISOnum */
  {"brvbar",  166}, /* broken bar = broken vertical bar, U+00A6 ISOnum */
  {"sect",    167}, /* section sign, U+00A7 ISOnum */
  {"uml",     168}, /* diaeresis = spacing diaeresis, U+00A8 ISOdia */
```

```
{"copy",    169}, /* copyright sign, U+00A9 ISOnum */
{"ordf",    170}, /* feminine ordinal indicator, U+00AA ISOnum */
{"laquo",   171}, /* left-pointing double angle quotation mark = left
                     pointing guillemet, U+00AB ISOnum */
{"not",     172}, /* not sign, U+00AC ISOnum */
{"shy",     173}, /* soft hyphen = discretionary hyphen, U+00AD ISOnum */
{"reg",     174}, /* registered sign = registered trade mark sign,
                    U+00AE ISOnum */
{"macr",    175}, /* macron = spacing macron = overline = APL overbar,
                    U+00AF ISOdia */
{"deg",     176}, /* degree sign, U+00B0 ISOnum */
{"plusmn",  177}, /* plus-minus sign = plus-or-minus sign, U+00B1 ISOnum */
{"sup2",    178}, /* superscript two = superscript digit two =
                    squared, U+00B2 ISOnum */
{"sup3",    179}, /* superscript three = superscript digit three =
                    cubed, U+00B3 ISOnum */
{"acute",   180}, /* acute accent = spacing acute,
                                  U+00B4 ISOdia */
{"micro",   181}, /* micro sign, U+00B5 ISOnum */
{"para",    182}, /* pilcrow sign = paragraph sign,
                                  U+00B6 ISOnum */
{"middot",  183}, /* middle dot = Georgian comma
                                  = Greek middle dot, U+00B7 ISOnum */
{"cedil",   184}, /* cedilla = spacing cedilla, U+00B8 ISOdia */
{"sup1",    185}, /* superscript one = superscript digit one,
                                  U+00B9 ISOnum */
{"ordm",    186}, /* masculine ordinal indicator,
                                  U+00BA ISOnum */
{"raquo",   187}, /* right-pointing double angle quotation mark
                                   = right pointing guillemet, U+00BB ISOnum */
{"frac14",  188}, /* vulgar fraction one quarter
                                  = fraction one quarter, U+00BC ISOnum */
{"frac12",  189}, /* vulgar fraction one half
                                  = fraction one half, U+00BD ISOnum */
{"frac34",  190}, /* vulgar fraction three quarters
                                = fraction three quarters, U+00BE ISOnum */
{"iquest",  191}, /* inverted question mark
                                  = turned question mark, U+00BF ISOnum */
{"Agrave",  192}, /* latin capital letter A with grave
                                  = latin capital letter A grave,
                                  U+00C0 ISOlat1 */
{"Aacute",  193}, /* latin capital letter A with acute,
                                  U+00C1 ISOlat1 */
{"Acirc",   194}, /* latin capital letter A with circumflex,
                                  U+00C2 ISOlat1 */
{"Atilde",  195}, /* latin capital letter A with tilde,
                                  U+00C3 ISOlat1 */
{"Auml",    196}, /* latin capital letter A with diaeresis,
                                  U+00C4 ISOlat1 */
{"Aring",   197}, /* latin capital letter A with ring above
                                  = latin capital letter A ring,
                                  U+00C5 ISOlat1 */
{"AElig",   198}, /* latin capital letter AE
                                  = latin capital ligature AE,
                                  U+00C6 ISOlat1 */
{"Ccedil",  199}, /* latin capital letter C with cedilla,
                                  U+00C7 ISOlat1 */
{"Egrave",  200}, /* latin capital letter E with grave,
                                  U+00C8 ISOlat1 */
{"Eacute",  201}, /* latin capital letter E with acute,
                                  U+00C9 ISOlat1 */
{"Ecirc",   202}, /* latin capital letter E with circumflex,
                                  U+00CA ISOlat1 */
{"Euml",    203}, /* latin capital letter E with diaeresis,
                                  U+00CB ISOlat1 */
{"Igrave",  204}, /* latin capital letter I with grave,
                                  U+00CC ISOlat1 */
{"Iacute",  205}, /* latin capital letter I with acute,
                                  U+00CD ISOlat1 */
{"Icirc",   206}, /* latin capital letter I with circumflex,
                                  U+00CE ISOlat1 */
{"Iuml",    207}, /* latin capital letter I with diaeresis,
                                  U+00CF ISOlat1 */
{"ETH",     208}, /* latin capital letter ETH, U+00D0 ISOlat1 */
{"Ntilde",  209}, /* latin capital letter N with tilde,
                                  U+00D1 ISOlat1 */
{"Ograve",  210}, /* latin capital letter O with grave,
                                  U+00D2 ISOlat1 */
{"Oacute",  211}, /* latin capital letter O with acute,
                                  U+00D3 ISOlat1 */
{"Ocirc",   212}, /* latin capital letter O with circumflex,
                                  U+00D4 ISOlat1 */
{"Otilde",  213}, /* latin capital letter O with tilde,
                                  U+00D5 ISOlat1 */
{"Ouml",    214}, /* latin capital letter O with diaeresis,
                                  U+00D6 ISOlat1 */
{"times",   215}, /* multiplication sign, U+00D7 ISOnum */
{"Oslash",  216}, /* latin capital letter O with stroke
                                  = latin capital letter O slash,
                                  U+00D8 ISOlat1 */
{"Ugrave",  217}, /* latin capital letter U with grave,
                                  U+00D9 ISOlat1 */
{"Uacute",  218}, /* latin capital letter U with acute,
                                  U+00DA ISOlat1 */
{"Ucirc",   219}, /* latin capital letter U with circumflex,
                                  U+00DB ISOlat1 */
{"Uuml",    220}, /* latin capital letter U with diaeresis,
                                  U+00DC ISOlat1 */
{"Yacute",  221}, /* latin capital letter Y with acute,
                                  U+00DD ISOlat1 */
{"THORN",   222}, /* latin capital letter THORN,
                                  U+00DE ISOlat1 */
{"szlig",   223}, /* latin small letter sharp s = ess-zed,
                                  U+00DF ISOlat1 */
{"agrave",  224}, /* latin small letter a with grave
                                  = latin small letter a grave,
                                  U+00E0 ISOlat1 */
```

```
  {"aacute",  225}, /* latin small letter a with acute,
                                   U+00E1 ISOlat1 */
  {"acirc",   226}, /* latin small letter a with circumflex,
                                   U+00E2 ISOlat1 */
  {"atilde",  227}, /* latin small letter a with tilde,
                                   U+00E3 ISOlat1 */
  {"auml",    228}, /* latin small letter a with diaeresis,
                                   U+00E4 ISOlat1 */
  {"aring",   229}, /* latin small letter a with ring above
                                   = latin small letter a ring,
                                   U+00E5 ISOlat1 */
  {"aelig",   230}, /* latin small letter ae
                               = latin small ligature ae, U+00E6 ISOlat1 */
  {"ccedil",  231}, /* latin small letter c with cedilla,
                                   U+00E7 ISOlat1 */
  {"egrave",  232}, /* latin small letter e with grave,
                                   U+00E8 ISOlat1 */
  {"eacute",  233}, /* latin small letter e with acute,
                                   U+00E9 ISOlat1 */
  {"ecirc",   234}, /* latin small letter e with circumflex,
                                   U+00EA ISOlat1 */
  {"euml",    235}, /* latin small letter e with diaeresis,
                                   U+00EB ISOlat1 */
  {"igrave",  236}, /* latin small letter i with grave,
                                   U+00EC ISOlat1 */
  {"iacute",  237}, /* latin small letter i with acute,
                                   U+00ED ISOlat1 */
  {"icirc",   238}, /* latin small letter i with circumflex,
                                   U+00EE ISOlat1 */
  {"iuml",    239}, /* latin small letter i with diaeresis,
                                   U+00EF ISOlat1 */
  {"eth",     240}, /* latin small letter eth, U+00F0 ISOlat1 */
  {"ntilde",  241}, /* latin small letter n with tilde,
                                   U+00F1 ISOlat1 */
  {"ograve",  242}, /* latin small letter o with grave,
                                   U+00F2 ISOlat1 */
  {"oacute",  243}, /* latin small letter o with acute,
                                   U+00F3 ISOlat1 */
  {"ocirc",   244}, /* latin small letter o with circumflex,
                                   U+00F4 ISOlat1 */
  {"otilde",  245}, /* latin small letter o with tilde,
                                   U+00F5 ISOlat1 */
  {"ouml",    246}, /* latin small letter o with diaeresis,
                                   U+00F6 ISOlat1 */
  {"divide",  247}, /* division sign, U+00F7 ISOnum */
  {"oslash",  248}, /* latin small letter o with stroke,
                                   = latin small letter o slash,
                                   U+00F8 ISOlat1 */
  {"ugrave",  249}, /* latin small letter u with grave,
                                   U+00F9 ISOlat1 */
  {"uacute",  250}, /* latin small letter u with acute,
                                   U+00FA ISOlat1 */
  {"ucirc",   251}, /* latin small letter u with circumflex,
                                   U+00FB ISOlat1 */
  {"uuml",    252}, /* latin small letter u with diaeresis,
                                   U+00FC ISOlat1 */
  {"yacute",  253}, /* latin small letter y with acute,
                                   U+00FD ISOlat1 */
  {"thorn",   254}, /* latin small letter thorn with, U+00FE ISOlat1 */
  {"yuml",    255}, /* latin small letter y with diaeresis, U+00FF ISOlat1 */
  {(char *)NULL     , 0}
};

/* Search the list of entities, and return the equivalent character - or
 * zero if none can be found
 */

char EntitiesSearch(char *name)
{
  int i=0;

  /* first we do a case sensitive search, there are subtle differences in
   * entity naming depending on the case of the appropriate output character
   */

  while(EntityNames[i].name != NULL)
  {
    if(!strcmp(EntityNames[i].name,name))
      return EntityNames[i].value;
    ++i;
  }

  i=0;

  /* we haven't found a match, so try a case insensitive search */
  while(EntityNames[i].name != NULL)
  {
    if(!stricmp(EntityNames[i].name,name))
      return EntityNames[i].value;
    ++i;
  }

  return 0;
}

/* Takes a block of text, processes all the entities within it, and returns
 * that block - returns the new length of that block
 *
 * Note that the code works on THE BLOCK THAT HAS BEEN PASSED. To preserve
 * the original pass a copy.
 *
 * returns: new length of block after entity processing
 *
 */
int EntitiesDeEntitise(char *block, int length)
{
  char *end;
  char *startent; /* start position of entity */
  char *endent;   /* end postition of entity  */
  char translated=0;

  end=block+length;

  while(block<=end)
  {
    while(*block!='&' && block!=end)
      ++block;

    printf("%s",block);

    if(*block=='&')
    {
      startent=block;

      if(*(block+1)=='#')
      {
        char number[8];
        int index=0;

        /* this is a numeric thingy
     * note that we allow numbers terminated with space instead of ;
         * bad HTML, but common :-(
         */
        while(index!=7 && *(2+block+index)!='\0' && *(2+block+index)!=';'
              && !isspace(*(2+block+index)) && !iscntrl(*(2+block+index)))
        {
          number[index]=*(2+block+index);
          number[index+1]='\0';
          ++index;
        }

        endent=2+block+index;

        if((atoi(number) < 255) && (atoi(number) > 1))
        {
          /* we can decode this ok */
          translated=atoi(number);
        }
      } else {
        char name[16];
        int index=0;

        /* this is a named attribute */
        while(index!=15 && *(1+block+index)!='\0' && *(1+block+index)!=';'
              && !isspace(*(1+block+index)) && !iscntrl(*(1+block+index)))
        {
          name[index]=*(1+block+index);
          name[index+1]='\0';
          ++index;
        }

        endent=(1+block+index);

        translated=EntitiesSearch(name);
      }
    }

    /* now we replace the area from & to ; with the character "translated"
     * ie                 &amp;
     *           startent ^   ^ endent
     *
     */
    if(translated!=0)
    {
       *startent=translated;

       printf("Translated=%c\n",translated);

       printf("endent=%i startent=%i end=%i end-endent=%i\n",
              (int)endent,(int)startent,(int)end,(int)(end-endent));

       printf("*endent=%c *startent=%c\n",*endent,*startent);
       printf("%s\n",block);

       /* now move the whole string back so :
        *   foobar flibble &amp; flirble
        * becomes :
        *   foobar flibble & flirble
        */
       memmove(startent+1,endent+1,(end-endent));

       length=length-(endent-startent);
       end=end-(endent-startent);
       block=startent;
       ++block;
    } else {
      ++block;
    }
    startent=NULL;
    endent=NULL;
    translated=0;
  }

  return length;
}

/* Takes a block of text, looks for entitisable characters within it, and
 * returns a new block, with entities - length is an
 * integer containing the source length, newlength is a pointer to
 * an integer to store the entitised length
 */
char *EntitiesEntitise(char *block, int length, int *newlength)
{
  int index;
  int tlen;
  char *outstring;

  *newlength=0;

  /* skip through and establish the length of the entitised text */
  index=0;
  while(index!=length)
  {
    int eindex=0;
    while(EntityNames[eindex].name != NULL)
    {
      if(EntityNames[eindex].value==block[index])
      {
        tlen=tlen+strlen(EntityNames[eindex].name)+1;
        break;
      }
      ++eindex;
    }
    ++tlen;
    ++index;
  }

  /* allocate memory for output */
  outstring=(char *)malloc(tlen+1);
  if(!outstring)
  {
    fprintf(stderr,"EntitiesEntitise: Out of memory (%i bytes needed)\n",
            tlen);
    return NULL;
  }

  strcpy(outstring,"");

  /* now do the copy */
  index=0;
  while(index<=length)
  {
    int eindex=0;

    printf("(%c)",block[index]);

    if(block[index]==0)
    {
      ++index;
      continue;
    }

    while(EntityNames[eindex].name != NULL)
    {
      if(EntityNames[eindex].value==block[index])
      {
        strcat(outstring,"&");
        strcat(outstring,EntityNames[eindex].name);
        strcat(outstring,";");
        break;
      }
      ++eindex;
    }

    /* if no entity was found just copy the character (if it's a valid
     * one)
     */
    if(EntityNames[eindex].name == NULL)
    {
      if(isalpha(block[index]) || ispunct(block[index]) ||
         isspace(block[index]))
      {
        outstring[strlen(outstring)+1]='\0';
        outstring[strlen(outstring)]=block[index];
      } else {
        char temp[6];

        sprintf(temp,"&#%03i;",(int)block[index]);
        strcat(outstring,temp);
      }
    }

    ++index;
  }

  *newlength=strlen(outstring);
  return outstring;
}

#ifdef ENTITIESTEST

/* simple test routine */
int main(void)
{
   char *testdata, *entitised;
   int testlength;
   int error;
   int newlength, length;

   setbuf(stdout,NULL);

   testdata=load_malloc("/home/thowat/project/test/entitytest.html",
                        &testlength,&error);

   switch(error)
   {
     case uOPENFAIL:
       fprintf(stderr,"No entity test file\n");
       exit(1);
       break;
     case uNOMEMORY:
       fprintf(stderr,"No memory for entity test file! Quitting.\n");
       exit(1);
       break;
   }

   testdata[testlength]='\0';
   printf("%s\n",testdata);
   newlength=EntitiesDeEntitise(testdata,testlength);
   testdata[newlength]='\0';
   printf("%s\n",testdata);

   entitised=EntitiesEntitise(testdata,newlength,&length);

   entitised[length]=0;

   if(entitised)
     printf("%s\n",entitised);

   return 0;
}

#endif
```

F.10 entities.h

```
/*
 * entities.h
 * Tony Howat, 1999/2000
 *
 * This is a simple piece of code to use a standard HTML 2.0 entity list
 * to translate blocks of text with HTML entities with them.
 *
 */

/* Search the list of entities, and return the equivalent character - or
 * zero if none can be found
 */

char EntitiesSearch(char *name);

/* Takes a block of text, processes all the entities within it, and returns
 * that block - returns the new length of that block
 *
 * Note that the code works on THE BLOCK THAT HAS BEEN PASSED. To preserve
 * the original pass a copy.
 *
 * returns: new length of block after entity processing
 *
 */
int EntitiesDeEntitise(char *block, int length);

/* Takes a block of text, looks for entitisable characters within it, and
 * returns a new block, with entities - length is an
 * integer containing the source length, newlength is a pointer to
 * an integer to store the entitised length
 */
char *EntitiesEntitise(char *block, int length, int *newlength);
```

F.11 fetch.c

```
/*
 * fetch.c
 * Tony Howat 2000
 *
 * The CGI front end for the search engine
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <time.h>
#include "util.h"
#include "global.h"
#include "stopwords.h"
#include "porter.h"
#include "debug.h"
#include "sni.h"
#include "snic.h"
#include "preprocdoc.h"
#include "chunks.h"

char header[]="/home/httpd/html/fetch/header.html";
char footer[]="/home/httpd/html/fetch/footer.html";

#define strcasecmp(a,b) stricmp(a,b)
#define server_error printf

char *findcgivar(char **cgivars,char *tag);
```

```
static void getcgivars_report_error(char *fmt, ...)
{
   va_list va;

   printf("<HTML><HEAD><TITLE>500 Internal Server Error</TITLE></HEAD>\n"
          "<BODY><H2>500 Internal Server Error</H2>\n");
   va_start(va,fmt);
   vprintf(fmt,va);
   va_end(va);
   printf("</BODY></HTML>");
}


/* copyfile: does exactly what it says on the tin
 */

void copyfile(char *file)
{
  FILE *in;

  in=fopen(file,"rb");
  if(in==NULL)
  {
      printf("Failed to open %s",file);
      exit(1);
  }

  while(!feof(in))
  {
      char c;

      c=fgetc(in);
      if(!feof(in))
          fputc(c,stdout);
  }

  fclose(in);
}

/* this routine will copy a list of cgi variables into hidden HTML
 * form tags, so states etc can be preserved
 */

void copycgivars(char **cgivars)
{
  int i=0;

  if(cgivars!=NULL)
  {
    for (i=0; cgivars[i]; i+= 2)
    {
      if(strcmp(cgivars[i+1],"") && strcmp(cgivars[i+1],"0"))
        printf("<input type=hidden name=%s value=\"%s\">\n",
               cgivars[i],cgivars[i+1]);
    }
  }
}

/*************************************************************************/
/**                                                                     **/
/**     getcgivars.c-- routine to read CGI input variables into an      **/
/**         array of strings.                                           **/
/**                                                                     **/
/**     The x2c() and unescape_url() routines were lifted directly      **/
/**     from NCSA's sample program util.c, packaged with their HTTPD.   **/
/**                                                                     **/
/*************************************************************************/

/** Convert a two-char hex string into the char it represents **/
char x2c(char *what) {
   register char digit;

   digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] - '0'));
   digit *= 16;
   digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1] - '0'));
   return(digit);
}

/** Reduce any %xx escape sequences to the characters they represent **/
void unescape_url(char *url) {
    register int i,j;

    for(i=0,j=0; url[j]; ++i,++j) {
        if((url[i] = url[j]) == '%') {
            url[i] = x2c(&url[j+1]) ;
            j+= 2 ;
        }
    }
    url[i] = '\0' ;
}

/** Read the CGI input and place all name/val pairs into list.        **/
/** Returns list containing name1, value1, name2, value2, ... , NULL  **/
char **getcgivars() {
    register int i ;
    char *request_method ;
    int content_length;
    char *cgiinput ;
    char **cgivars ;
    char **pairlist ;
    int paircount ;
    char *nvpair ;
    char *eqpos ;

    /** Depending on the request method, read all CGI input into cgiinput **/
    /** (really should produce HTML error messages, instead of exit()ing) **/
    request_method= getenv("REQUEST_METHOD") ;

    if(!request_method)
      return NULL;

    if (!strcmp(request_method, "GET") || !strcmp(request_method, "HEAD") ) {
       if(getenv("QUERY_STRING"))
           cgiinput= strdup(getenv("QUERY_STRING"));
       else
           return NULL;
    }
    else if (!strcmp(request_method, "POST")) {
        /* strcasecmp() is not supported in Windows-- use strcmpi() instead */
        if(!getenv("CONTENT_TYPE"))
        {
            getcgivars_report_error("getcgivars(): Unsupported Content-Type.\n") ;
            exit(1) ;
        }

        if ( strcasecmp(getenv("CONTENT_TYPE"), "application/x-www-form-urlencoded")) {
            getcgivars_report_error("getcgivars(): Unsupported Content-Type.\n") ;
            exit(1) ;
        }
        if ( !(content_length = atoi(getenv("CONTENT_LENGTH"))) ) {
            getcgivars_report_error("getcgivars(): No Content-Length was sent with the POST request.\n") ;
            exit(1) ;
        }
        if ( !(cgiinput= (char *) malloc(content_length+1)) ) {
            getcgivars_report_error("getcgivars(): Could not malloc for cgiinput.\n") ;
            exit(1) ;
        }
        if (!fread(cgiinput, content_length, 1, stdin)) {
            getcgivars_report_error("Couldn't read CGI input from STDIN.\n") ;
            exit(1) ;
        }
        cgiinput[content_length]='\0' ;

    }
    else {
        getcgivars_report_error("getcgivars(): unsupported REQUEST_METHOD\n") ;
        exit(1) ;
    }

    /** Change all plusses back to spaces **/
    for(i=0; cgiinput[i]; i++) if(cgiinput[i] == '+') cgiinput[i] = ' ' ;

    /** First, split on "&" to extract the name-value pairs into pairlist **/
    pairlist= (char **) malloc(256*sizeof(char **)) ;
    paircount= 0 ;
    nvpair= strtok(cgiinput, "&") ;
    while (nvpair) {
        pairlist[paircount++]= strdup(nvpair) ;
        if (!(paircount%256))
            pairlist= (char **) realloc(pairlist,(paircount+256)*sizeof(char **)) ;
        nvpair= strtok(NULL, "&") ;
    }
    pairlist[paircount]= 0 ;    /* terminate the list with NULL */

    /** Then, from the list of pairs, extract the names and values **/
    cgivars= (char **) malloc((paircount*2+1)*sizeof(char **)) ;
    for (i= 0; i<paircount; i++) {
        if (eqpos=strchr(pairlist[i], '=')) {
            *eqpos= '\0' ;
            unescape_url(cgivars[i*2+1]= strdup(eqpos+1)) ;
        } else {
            unescape_url(cgivars[i*2+1]= strdup("")) ;
        }
        unescape_url(cgivars[i*2]= strdup(pairlist[i])) ;
    }
    cgivars[paircount*2]= 0 ;   /* terminate the list with NULL */

    /** Free anything that needs to be freed **/
    free(cgiinput) ;
    for (i=0; pairlist[i]; i++) free(pairlist[i]) ;
    free(pairlist) ;

    /** Return the list of name-value strings **/
    return cgivars ;

}

/* look up string value of a cgi variable */

char *findcgivar(char **cgivars,char *tag)
{
  int i=0;

  if(cgivars!=NULL)
  {
    for (i=0; cgivars[i]; i+= 2)
      if(!stricmp(cgivars[i],tag))
      {
        return cgivars[i+1];
      }
  }

  return NULL;
}

/* format a result into HTML */
#ifdef DEBUG
static void print_search_result(char *title, char *url, char *abstract,
                                int length, time_t t, int weight)
#else
static void print_search_result(char *title, char *url, char *abstract,
                                int length, time_t t)
#endif
```

```
{
  char *cp;
  char *urlt;

  cp=ctime(&t);
  cp[strlen(cp)-1]='\0'; /* get rid of \n */

  /* TEMPORARY - this fixes locally sourced test pages to proper URLs */
  if(!strncmp(url,"/export/wgot/",13))
    urlt=url+13;
  else {
    if(!strncmp(url,"/export/wgot2/",14) || !strncmp(url,"/export/wgot3/",14))
      urlt=url+14;
    else
      urlt=url;
  }

#ifdef DEBUG

  printf("<p><A HREF=\"http://%s\">%s</A> <font color=red>(Weight %i)</font>"
         " <font size=-1><br>%s...<br>\n"
         "<A HREF=\"http://%s\">http://%s</A> - <font color=green>%i"
         " bytes, fetched on %s</font></font>\n",urlt,title,weight,
         abstract,urlt,urlt,length,cp);
#else
  printf("<p><A HREF=\"http://%s\">%s</A><font size=-1><br>%s...<br>\n"
         "<A HREF=\"http://%s\">http://%s</A> - <font color=green>%i"
         " bytes, fetched on %s</font></font>\n",urlt,title,abstract,urlt,
         urlt,length,cp);
#endif
}


/* fetch a document's details and output it as a search result
 * will use chunk_id instead if c is non-zero
 */
#ifndef DEBUG
void print_document_data(int hash, chunk_id c)
#else
void print_document_data(int hash, chunk_id c, int weight)
#endif
{
  sni_response sr;
  DocumentRecord *dr;
  char *url;

  if(c)
    sni_send(SNISERVER, SNIPORT, SNI_GETDOCCHUNK, (void *)&c, 4, &sr);
  else
    sni_send(SNISERVER, SNIPORT, SNI_GETDOCRECORD, (void *)&hash, 4, &sr);

  if(sr.type!=SNI_DOCRECORD | sr.length==0)
  {
    /* this should be mailed to admin too! */
    if(hash)
    {
      printf("<p><font color=red>Failed to find docrecord with hash 0x%x - "
             "reponse 0x%x report to administrator!</font><br>\n",
             hash,sr.type);
    } else {
      printf("<p><font color=red>Failed to find docrecord at chunk 0x%x - "
             "reponse 0x%x report to administrator!</font><br>\n",
             (int)c,sr.type);
    }

    if(sr.data!=NULL)
      free(sr.data);
    return;
  }

  /* get pointer to DocumentRecord */
  dr=(DocumentRecord *)sr.data;
  url=(char *)dr+(sizeof(DocumentRecord)); /* find where url is */

  /* output it */
#ifndef DEBUG
  print_search_result(dr->title, url, dr->abstract, dr->length, dr->time);
#else
  print_search_result(dr->title, url, dr->abstract, dr->length, dr->time,
                      weight);
#endif

  free(sr.data);
}

/***************** end of the getcgivars() module ********************/

/* take a query string, process it and output any results */
void do_query(char *q)
{
  char *ptr;
  char *startpointer=NULL; /* start of term pointer */
  char *queryblock=NULL;
  int  terms=0;
  int  *termptr;
  sni_response sr; /* structure for storing net response data */

  /* first take the query and split it into seperate terms */

  ptr=q;

  /* allocate enough memory for term count (integer) */
  queryblock=malloc(sizeof(int));
  if(queryblock==NULL)
  {
    /* this, of course is a server problem from the user's
     * perspective, not a problem with our database server */
    printf("Server problem! - no memory for query!<br>\n");
    return;
  }

  while(*ptr!='\0')
  {
    /* if we have no start term reference, and ptr is at
     * a suitable letter, then use it as start of term
     */
    if(isalpha(*ptr))
    {
      char word[TERMSTRINGLENGTH];
      char *wordp;
      int windex=0;

      startpointer=ptr;
      while(*ptr!='\0' && isalpha(*ptr))
        ++ptr;
      while(windex<TERMSTRINGLENGTH && startpointer+windex!=ptr)
      {
        word[windex]=tolower(startpointer[windex]);
        ++windex;
      }
      word[windex]='\0';

      wordp=strippunct(word);

      /* now, as per preprocdochtml we see if this term is suitable
       * if so include it in our query structure
       */
      if(IsValidTerm(wordp))
      {
        char *t;

        queryblock=realloc(queryblock,
                           ((terms+1)*TERMSTRINGLENGTH)+sizeof(int));
        if(queryblock==NULL)
    {
          printf("Server problem! - no memory for query!\n<br>");
          return;
        }
        /* find a pointer to string area just allocated */
        t=queryblock+((TERMSTRINGLENGTH*terms)+sizeof(int));
        memset(t,'\0',TERMSTRINGLENGTH);  /* clear block */
        strcpy(t,wordp);                  /* fill it in */
        ++terms;
      }
    }

    ++ptr;
  }

  termptr=(int *)queryblock;
  *termptr=terms;

  printf("Search for terms : <font color=green>");

  terms=0;

  while(terms!=*termptr)
  {
    char *wp;

    /* find a pointer to the word/term */
    wp=queryblock+sizeof(int)+(TERMSTRINGLENGTH*terms);

    printf("%s",wp);
#ifdef STEMMING
    /* now use Porter's stemming algorithm on the word */
    wp[(stem(wp,0,strlen(wp)-1))+1]='\0';
#endif
    ++terms;
    if(terms!=*termptr)
      printf(", ");
  }

#ifdef STEMMING

  printf("</font> (<font color=blue>");

  terms=0;
  while(terms!=*termptr)
  {
    printf("%s",queryblock+sizeof(int)+(TERMSTRINGLENGTH*terms));
    ++terms;
    if(terms!=*termptr)
      printf(", ");
  }

#endif

  printf("</font>)<br>\n");

#ifdef DUMPDATA
  printf("<pre>");
  debug_dumper(queryblock,sizeof(int)+(TERMSTRINGLENGTH*terms),
               stdout,DUMP_WORD);
  printf("</pre>");
#endif

  /* send query to server */
  sni_send(SNISERVER, SNIPORT, SNI_QUERY, (void *)queryblock,
           sizeof(int)+(TERMSTRINGLENGTH*terms), &sr);

#ifdef DUMPDATA
  printf("<pre>Response type %i (0x%x)<br>",sr.type,sr.type);
  debug_dumper(sr.data,sr.length,stdout,DUMP_WORD);
  printf("</pre><p>");
#endif

  if(sr.type!=SNI_QUERYRESULTS)
  {
    printf("<p><font color=red>Query failed or was rejected, sorry. "
           "You may like to try again later.</font><br>\n",sr.type);
    if(sr.data!=NULL)
      free(sr.data);
    return;
  } else {
    int *results;

    printf("<font color=green>%i matches to query.</font><p>\n",
           sr.length/(sizeof(int)*2));

    results=(int *)sr.data;

    while(results!=(int *)(sr.data+sr.length))
    {
#ifdef DEBUG
      print_document_data(0,*results,*(results+1));
#else
      print_document_data(0,*results,*(results+1));
#endif
      results+=2;
    }
  }

  free(queryblock);

  if(sr.data!=NULL)
    free(sr.data);
  return;
}

/* the main routine */
int main(void)
{
  char **cgivars ;
  int i ;

  load_stopwords("/home/thowat/project/stopwords");

  setbuf(stdout,NULL);

  /** Print the CGI response header, required for all HTML output. **/
  /** Note the extra \n, to send the blank line.                  **/
  printf("Content-type: text/html\n\n") ;

  /** First, get the CGI variables into a list of strings         **/
  cgivars= getcgivars() ;

  copyfile(header);

  /* deal with an info request - different from a query */
  if(findcgivar(cgivars,"info"))
  {
    sni_response sr;

    sni_send(SNISERVER, SNIPORT, SNI_MSGINFO, "info", 4, &sr);

    if(sr.type!=SNI_OK | sr.length==0)
    {
      printf("<p><font color=red>Failed to fetch server information!"
             "</font><br>\n");
    } else {
      sr.data[sr.length]='\0';
      printf("<p><b>fetch CGI client ("__DATE__" " __TIME__")</b><br>%s\n",
             sr.data);
    }
    if(sr.data!=NULL)
      free(sr.data);
  }

  if(findcgivar(cgivars,"q"))
  {
    copycgivars(cgivars);
    do_query(findcgivar(cgivars,"q"));
  }

#ifdef dumpcgivars

    printf("<html>\n") ;
    printf("<head><title>CGI Results</title></head>\n") ;
    printf("<body>\n") ;
    printf("<h1>Hello, world.</h1>\n") ;
    printf("Your CGI input variables were:\n") ;
    printf("<ul>\n") ;

    /** Print the CGI variables sent by the user.  Note the list of **/
    /**   variables alternates names and values, and ends in NULL.  **/
    if(cgivars!=NULL)
    {
      for (i=0; cgivars[i]; i+= 2)
        printf("<li>[%s] = [%s]\n", cgivars[i], cgivars[i+1]) ;
    }

    printf("</ul>\n") ;
    printf("</body>\n") ;
    printf("</html>\n") ;

#endif

  copyfile(footer);

  /** Free anything that needs to be freed **/
  if(cgivars)
  {
    for (i=0; cgivars[i]; i++) free(cgivars[i]) ;
    free(cgivars) ;
  }

  free_stopwords();

  exit(0);
}
```

F.12 global.h

```
/*
 * global.h
 *
 * global compile time definitions
 *
 */

//#define DEBUG                1   /* diagnostics? */
#define VIRTUALTIME          1   /* use virtual timers */
//#define PROFILING            1   /* cd to profiling dir for child */
#define FILESTATS            1   /* dump statistics to disc */
#define FILESTATSTEP         50  /* dump stats every n submissions */
#define STEMMING             1   /* use porter's stemming algorithm? */
#define ABSLEN               132 /* max abstract length */
#define TITLELEN             64  /* max title string length */
#define MAXTERMS             20  /* number of terms to index per doc(max) */
#define URLCHUNKLENGTH       32  /* length of chunks to use for URLs */
#define WORDFILECHUNKLENGTH  64  /* length of chunks to use for words */
#define TERMSTRINGLENGTH     32  /* max length of a term string */

#define DBROOTFILE "/home/thowat/project/fetchdb"  /* where data should be */
#define PROFILEDIR "/home/thowat/project/cprof"
                                            /* where to put child profile */
#define STATSFILE  "/home/thowat/project/stats" /* where to put stats */

#if VIRTUALTIME && PROFILING
#error "VIRTUALTIME and PROFILING are mutually exclusive as they use the same timers, #undef one!"
#endif
```

F.13 porter.c

```
/*
 * porter.c
 *
 * Martin Porter's reference implementation of his stemming
 * algorithm, modified slightly.
 *
 */


/* This is the Porter stemming algorithm, coded up in ANSI C by the
   author. It may be be regarded as cononical, in that it follows the
   algorithm presented in

   Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,
   no. 3, pp 130-137,

   only differing from it at the points maked --DEPARTURE-- below.

   See also http://www.muscat.com/~martin/stem.html

   The algorithm as described in the paper could be exactly replicated
   by adjusting the points of DEPARTURE, but this is barely necessary,
   because (a) the points of DEPARTURE are definitely improvements, and
   (b) no encoding of the Porter stemmer I have seen is anything like
   as exact as this version, even with the points of DEPARTURE!

   You can compile it on Unix with 'gcc -O3 -o stem stem.c' after which
   'stem' takes a list of inputs and sends the stemmed equivalent to
   stdout.

   The algorithm as encoded here is particularly fast.

   Release 1
*/

#include <stdio.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

/* The main part of the stemming algorithm starts here. b is a buffer
   holding a word to be stemmed. The letters are in b[k0], b[k0+1] ...
   ending at b[k]. In fact k0 = 0 in this demo program. k is readjusted
   downwards as the stemming progresses. Zero termination is not in fact
   used in the algorithm.

   Note that only lower case sequences are stemmed. Forcing to lower case
   should be done before stem(...) is called.
*/

static char * b;       /* buffer for word to be stemmed */
static int k,k0,j;     /* j is a general offset into the string */

/* cons(i) is TRUE <=> b[i] is a consonant. */

static int cons(int i)
{  switch (b[i])
   {  case 'a': case 'e': case 'i': case 'o': case 'u': return FALSE;
      case 'y': return (i==k0) ? TRUE : !cons(i-1);
      default: return TRUE;
   }
}

/* m() measures the number of consonant sequences between k0 and j. if c is
   a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
   presence,

      <c><v>       gives 0
      <c>vc<v>     gives 1
      <c>vcvc<v>   gives 2
      <c>vcvcvc<v> gives 3
      ....
*/

static int m()
{  int n = 0;
   int i = k0;
   while(TRUE)
   {  if (i > j) return n;
      if (! cons(i)) break; i++;
   }
   i++;
   while(TRUE)
   {  while(TRUE)
      {  if (i > j) return n;
            if (cons(i)) break;
            i++;
      }
      i++;
      n++;
      while(TRUE)
      {  if (i > j) return n;
         if (! cons(i)) break;
         i++;
      }
      i++;
    }
}

/* vowelinstem() is TRUE <=> k0,...j contains a vowel */

static int vowelinstem()
{  int i; for (i = k0; i <= j; i++) if (! cons(i)) return TRUE;
   return FALSE;
}

/* doublec(j) is TRUE <=> j,(j-1) contain a double consonant. */

static int doublec(int j)
{  if (j < k0+1) return FALSE;
   if (b[j] != b[j-1]) return FALSE;
   return cons(j);
}

/* cvc(i) is TRUE <=> i-2,i-1,i has the form consonant - vowel - consonant
   and also if the second c is not w,x or y. this is used when trying to
   restore an e at the end of a short word. e.g.

      cav(e), lov(e), hop(e), crim(e), but
      snow, box, tray.

*/

static int cvc(int i)
{  if (i < k0+2 || !cons(i) || cons(i-1) || !cons(i-2)) return FALSE;
   {  int ch = b[i];
      if (ch == 'w' || ch == 'x' || ch == 'y') return FALSE;
   }
   return TRUE;
}

/* ends(s) is TRUE <=> k0,...k ends with the string s. */

static int ends(char * s)
{  int length = s[0];
   if (s[length] != b[k]) return FALSE; /* tiny speed-up */
   if (length > k-k0+1) return FALSE;
   if (memcmp(b+k-length+1,s+1,length) != 0) return FALSE;
   j = k-length;
   return TRUE;
}

/* setto(s) sets (j+1),...k to the characters in the string s, readjusting
   k. */

static void setto(char * s)
{  int length = s[0];
   memmove(b+j+1,s+1,length);
   k = j+length;
}

/* r(s) is used further down. */

static void r(char * s) { if (m() > 0) setto(s); }
```

```
/* step1ab() gets rid of plurals and -ed or -ing. e.g.

       caresses  ->  caress
       ponies    ->  poni
       ties      ->  ti
       caress    ->  caress
       cats      ->  cat

       feed      ->  feed
       agreed    ->  agree
       disabled  ->  disable

       matting   ->  mat
       mating    ->  mate
       meeting   ->  meet
       milling   ->  mill
       messing   ->  mess

       meetings  ->  meet

*/

static void step1ab()
{  if (b[k] == 's')
   {  if (ends("\04" "sses")) k -= 2; else
      if (ends("\03" "ies")) setto("\01" "i"); else
      if (b[k-1] != 's') k--;
   }
   if (ends("\03" "eed")) { if (m() > 0) k--; } else
   if ((ends("\02" "ed") || ends("\03" "ing")) && vowelinstem())
   {  k = j;
      if (ends("\02" "at")) setto("\03" "ate"); else
      if (ends("\02" "bl")) setto("\03" "ble"); else
      if (ends("\02" "iz")) setto("\03" "ize"); else
      if (doublec(k))
      {  k--;
         {  int ch = b[k];
            if (ch == 'l' || ch == 's' || ch == 'z') k++;
         }
      }
      else if (m() == 1 && cvc(k)) setto("\01" "e");
  }
}

/* step1c() turns terminal y to i when there is another vowel in the stem. */

static void step1c() { if (ends("\01" "y") && vowelinstem()) b[k] = 'i'; }


/* step2() maps double suffices to single ones. so -ization ( = -ize plus
   -ation) maps to -ize etc. note that the string before the suffix must give
   m() > 0. */

static void step2() { switch (b[k-1])
{
    case 'a': if (ends("\07" "ational")) { r("\03" "ate"); break; }
              if (ends("\06" "tional")) { r("\04" "tion"); break; }
              break;
    case 'c': if (ends("\04" "enci")) { r("\04" "ence"); break; }
              if (ends("\04" "anci")) { r("\04" "ance"); break; }
              break;
    case 'e': if (ends("\04" "izer")) { r("\03" "ize"); break; }
              break;
    case 'l': if (ends("\03" "bli")) { r("\03" "ble"); break; } /*-DEPARTURE-*/

 /* To match the published algorithm, replace this line with
    case 'l': if (ends("\04" "abli")) { r("\04" "able"); break; } */

              if (ends("\04" "alli")) { r("\02" "al"); break; }
              if (ends("\05" "entli")) { r("\03" "ent"); break; }
              if (ends("\03" "eli")) { r("\01" "e"); break; }
              if (ends("\05" "ousli")) { r("\03" "ous"); break; }
              break;
    case 'o': if (ends("\07" "ization")) { r("\03" "ize"); break; }
              if (ends("\05" "ation")) { r("\03" "ate"); break; }
              if (ends("\04" "ator")) { r("\03" "ate"); break; }
              break;
    case 's': if (ends("\05" "alism")) { r("\02" "al"); break; }
              if (ends("\07" "iveness")) { r("\03" "ive"); break; }
              if (ends("\07" "fulness")) { r("\03" "ful"); break; }
              if (ends("\07" "ousness")) { r("\03" "ous"); break; }
              break;
    case 't': if (ends("\05" "aliti")) { r("\02" "al"); break; }
              if (ends("\05" "iviti")) { r("\03" "ive"); break; }
              if (ends("\06" "biliti")) { r("\03" "ble"); break; }
              break;
    case 'g': if (ends("\04" "logi")) { r("\03" "log"); break; } /*-DEPARTURE-*/

 /* To match the published algorithm, delete this line */

} }

/* step3() deals with -ic-, -full, -ness etc. similar strategy to step2. */

static void step3() { switch (b[k])
{
    case 'e': if (ends("\05" "icate")) { r("\02" "ic"); break; }
              if (ends("\05" "ative")) { r("\00" ""); break; }
              if (ends("\05" "alize")) { r("\02" "al"); break; }
              break;
    case 'i': if (ends("\05" "iciti")) { r("\02" "ic"); break; }
              break;
    case 'l': if (ends("\04" "ical")) { r("\02" "ic"); break; }
              if (ends("\03" "ful")) { r("\00" ""); break; }
              break;
    case 's': if (ends("\04" "ness")) { r("\00" ""); break; }
```

```
              break;
} }

/* step4() takes off -ant, -ence etc., in context <c>vcvc<v>. */

static void step4()
{  switch (b[k-1])
    {  case 'a': if (ends("\02" "al")) break; return;
       case 'c': if (ends("\04" "ance")) break;
                 if (ends("\04" "ence")) break; return;
       case 'e': if (ends("\02" "er")) break; return;
       case 'i': if (ends("\02" "ic")) break; return;
       case 'l': if (ends("\04" "able")) break;
                 if (ends("\04" "ible")) break; return;
       case 'n': if (ends("\03" "ant")) break;
                 if (ends("\05" "ement")) break;
                 if (ends("\04" "ment")) break;
                 if (ends("\03" "ent")) break; return;
       case 'o': if (ends("\03" "ion") && (b[j] == 's' || b[j] == 't')) break;
                 if (ends("\02" "ou")) break; return;
                 /* takes care of -ous */
       case 's': if (ends("\03" "ism")) break; return;
       case 't': if (ends("\03" "ate")) break;
                 if (ends("\03" "iti")) break; return;
       case 'u': if (ends("\03" "ous")) break; return;
       case 'v': if (ends("\03" "ive")) break; return;
       case 'z': if (ends("\03" "ize")) break; return;
       default: return;
    }
    if (m() > 1) k = j;
}

/* step5() removes a final -e if m() > 1, and changes -ll to -l if
   m() > 1. */

static void step5()
{  j = k;
   if (b[k] == 'e')
   {  int a = m();
      if (a > 1 || a == 1 && !cvc(k-1)) k--;
   }
   if (b[k] == 'l' && doublec(k) && m() > 1) k--;
}

/* In stem(p,i,j), p is a char pointer, and the string to be stemmed is from
   p[i] to p[j] inclusive. Typically i is zero and j is the offset to the last
   character of a string, (p[j+1] == '\0'). The stemmer adjusts the
   characters p[i] ... p[j] and returns the new end-point of the string, k.
   Stemming never increases word length, so i <= k <= j. To turn the stemmer
   into a module, declare 'stem' as extern, and delete the remainder of this
   file.
*/

int stem(char * p, int i, int j)
{
   b = p; k = j; k0 = i; /* copy the parameters into statics */

   //printf("stem(\"%s\",%i,%i)",p,i,j);

   if (k <= k0+1)
   {
     //printf(" = %i\n",k);
      return k; /*-DEPARTURE-*/
   }

   /* With this line, strings of length 1 or 2 don't go through the
      stemming process, although no mention is made of this in the
      published algorithm. Remove the line to match the published
      algorithm. */

   step1ab(); step1c(); step2(); step3(); step4(); step5();

   //printf(" = %i\n",k);
   return k;
}

#ifdef ENDOF

/*--------------------stemmer definition ends here------------------------*/

static char * s;         /* a char * (=string) pointer; passed into b above */

#define INC 50           /* size units in which s is increased */
static int i_max = INC;  /* maximum offset in s */

void increase_s()
{  i_max += INC;
   {  char * new_s = (char *) malloc(i_max+1);
      { int i; for (i = 0; i < i_max; i++) new_s[i] = s[i]; } /* copy across */
      free(s); s = new_s;
   }
}

#define UC(ch) (ch <= 'Z' && ch >= 'A')
#define LC(ch) (ch <= 'z' && ch >= 'a')
#define LETTER(ch) (UC(ch) || LC(ch))
#define FORCELC(ch) (ch-('A'-'a'))

void stemfile(FILE * f)
{  while(TRUE)
   {  int ch = getc(f);
      if (ch == EOF) return;
      if (LETTER(ch))
      {  int i = 0;
         while(TRUE)
         {  if (i == i_max) increase_s();
```

```
            if UC(ch) ch = FORCELC(ch);
            /* forces lower case. Remove this line to make the program work
               exactly like the Muscat stemtext command. */

            s[i] = ch; i++;
            ch = getc(f);
            if (!LETTER(ch)) { ungetc(ch,f); break; }
         }
         s[stem(s,0,i-1)+1] = 0;
         /* the pevious line calls the stemmer and uses its result to
            zero-terminate the string in s */
         printf("%s",s);
      }
      else putchar(ch);
   }
}

int main(int argc, char * argv[])
{  int i;
   s = (char *) malloc(i_max+1);
   for (i = 1; i < argc; i++)
   {  FILE * f = fopen(argv[i],"r");
      if (f == 0) { fprintf(stderr,"File %s not found\n",argv[i]); exit(1); }
      stemfile(f);
   }
   free(s);
   return 0;
}

#endif
```

F.14 porter.h

```
/*
 * porter.c
 *
 * Martin Porter's reference implementation of his stemming
 * algorithm, modified slightly.
 *
 */

/* In stem(p,i,j), p is a char pointer, and the string to be stemmed is from
   p[i] to p[j] inclusive. Typically i is zero and j is the offset to the last
   character of a string, (p[j+1] == '\0'). The stemmer adjusts the
   characters p[i] ... p[j] and returns the new end-point of the string, k.
   Stemming never increases word length, so i <= k <= j.
*/

int stem(char * p, int i, int j);
```

F.15 preprocdoc.h

```
/*
 * preprocdoc.h
 *
 * Internal format for storing details extracted from a document. Data
 * may or may not be filled in depending on source document type.
 *
 */

#ifndef __PREPROCDOC_H
#define __PREPROCDOC_H 1

#include

typedef struct
{
  char word[32];
  int occurrences;
  int weighting;
} WordEntry;

/* A PreProcDoc structure is sent to the data server when a document has
 * been analysed. This structure is followed by PreProcDoc.indexedwords
 * WordEntry elements. At the end of this ( offset =
 * sizeof(PreProcDoc)+(ppd.indexedwords*sizeof(WordEntry)) ) there is
 * a string, which contains the url - these pointers have to be obtained
 * manually as C types don't like variable size fields!
 */

typedef struct
{
  char title[80];     /* document title */
  char abstract[132]; /* document abstract */
  time_t time;        /* time of indexing */
  int doclength;      /* original document length */
  int wordcount;      /* count of words indexed */
  int indexedwords;   /* count of weight/word pairs to add to index */
  /* WordEntry wlist;  * there follows a list of [indexedwords] WordEntry
                       * elements */
  /* char *url[]       * and then the source url */
} PreProcDoc;

typedef struct
{
  int  urlhash;      /* hash of lowest form url */
  int  sourceurl;    /* chunk pointer to url in urls file - stored like this
```

```
                     * as urls vary drastically in length */
  char title[80];    /* document title */
  char abstract[132];/* first 131 bytes of text, abstract, whatever */
  time_t time;       /* time document was indexed */
  int  length;       /* length of original document in bytes */
} DocumentRecord;

#endif
```

F.16 preprochtml.c

```
/*
 * preprochtml
 * Tony Howat 1999/2000
 *
 * Loads a HTML document, stripping tags which are irrelevant to us,
 * and preserving relevant tags in an internal (more useable) form.
 * Stores all META tags, and other document information
 *
 * test using find /export/wgot | grep .html | xargs -n 1 ./preprochtml >> out
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include "util.h"
#include "tags.h"
#include "porter.h"
#include "entities.h"
#include "preprocdoc.h"
#include "global.h"
#include "sni.h"
#include "snic.h"
#include "debug.h"
#include "stopwords.h"

#define STEMMING 1

#define WORDLIST_GRANULARITY 64  /* allocate memory for word tree in
                                  * blocks of... */

/* this defines the bitfield used to represent the current state in a
 * html stream
 */
typedef enum {
  tag_body  = 0x1,
  tag_title = 0x2,
  tag_h1    = 0x4,
  tag_h2    = 0x8,
  tag_h3    = 0x10,
  tag_h4    = 0x20,
  tag_h5    = 0x40,
  tag_h6    = 0x80,
  tag_bold  = 0x100
} tagstate;

static char *tagstate_names[]={"body","title","h1","h2","h3","h4","h5","h6",
                               "b",NULL};

/*typedef struct
{
  char tag[12];
  int weighting;

} TagDefinition;

static const TagDefinition TagTable[] = {
{
  { "H1",    10 }
  { "H2",     9 }
  { "H3",     8 }
  { "H4",     7 }
  { "H5",     6 }
  { "H6",     5 }
};*/

/*
 * Meta tags are stored as in a list, one MetaElement for each Meta tag.
 * The contents/parameters of each meta element are stored in a list
 * of MetaElementAttributes. ie :
 *
 * <META name="foo" content="baa" scheme="moo" http-equiv="woo">
 *  ^                ^      ^    ^      ^     ^
 *  |                |      |    |      |     |
 *  metaelement.name |      |    |      |     metaelement.httpequiv
 *                   |      |    |      metaelement.metaelementattribute[1]
 *                   |      |    |                                .content
 *                   |      |    metaelement.metaelementattribute[1].name
 *                   |      metaelement.metaelementattribute[0].content
 *                   metaelement.metaelementattribute[0].name
 */

typedef struct
{
  char *name;        /* an argument in a HTML meta tag */
  char *content;
} MetaElementAttribute;
```

```
typedef struct
{
  char *name;         /* contents of a HTML meta tag */
  char *httpequiv;
  MetaElementAttribute *attributes;
} MetaElement;

/*
 * A WordRecord is created for every word in the body text of a document.
 * Each word is stored in lower case, with upper case characters noted in
 * a 32 bit field. (This allows quicker word based string compares during
 * normal operation). CRC of the word is used as a hash of each record,
 * and using the CRC the table of words is organised into a B tree.
 * WordRecords are allocated in chunks of WORDLIST_GRANULARITY, in order
 * to reduce the number of calls to malloc whilst processing.
 */

typedef struct
{
  char word[32];       /* word these stats are for */
  int  ucasefield;     /* a bit is set for every upper case char in
                        * original word */
  int  occurrences;    /* number of occurences */
  int weighting;       /* any additional weightings */
  int  crc;            /* crc of word for faster searches */
  int  left;           /* binary tree left (less than) */
  int  right;          /* binary tree right (greater than or equal) */
} WordRecord;

typedef struct
{
  char abstract[ABSLEN];  /* first (ABSLEN) bytes of body text, or summary */
  char title[TITLELEN];   /* first (TITLELEN) bytes of page title */
  char *url;              /* source URL in lowest form */
  int abstractoffset;     /* next byte to fill in abstract offset */
  int trueabstract;       /* set to 1 if a true abstract has been extracted
                           * i.e. we shouldn't fill rest with body text */
  int titleoffset;        /* next byte to fill in title offset */
  WordRecord *wordlist;   /* word by word data */
  int wordlistnext;       /* index of next word to add - also a count
             * of words in document */
  int weightings;         /* total extra weightings */
  int follow;             /* true if we should follow links */
  int index;              /* index words in this page */
  int doclength;          /* original document length */
  int terms;              /* the final number of terms to index */
  time_t time;            /* time of index */
} PreProcessedDocument;

/* simple enum type used to store which direction a branch should be added
 * in
 */
typedef enum _branchdir
{
  BranchUnknown = 0,
  BranchRight,
  BranchLeft
} branchdir;

/*
 *  static int checktag(tagstate *ts,char *tag)
 *
 *  This routine updates a tagstate field according to the HTML tag
 *  passed. ie, passing "/body" will unset the body flag in the
 *  tagstate field pointed to by ts
 *
 *  returns TRUE if it processed the tag, FALSE if ignored
 *
 */

static int checktag(tagstate *ts,char *tag)
{
  int on=1; /* set to 1 if tag is being turned on */
  int index=0;

  if(*tag=='/') /* tag is being turned off */
  {
    on=0;
    tag++;
  }

  while(tagstate_names[index]!=NULL)
  {
    //printf("strincmp %s %s %i\n",tagstate_names[index],tag,strlen(tagstate_names[index]));
    if(!strincmp(tagstate_names[index],tag,strlen(tagstate_names[index])))
    {
      //printf("matched %i\n",tag[strlen(tagstate_names[index])-1]);
      /* we may have matched B to BODY incorrectly, check that the
       * following character is a whitespace or the null terminator
       */

      if(tag[strlen(tagstate_names[index])]=='\0' ||
         isspace(tag[strlen(tagstate_names[index])]))
      {
        //printf("%s/%s %s\n",tagstate_names[index],tag,on ? "on" : "off");

        /* this tag matches, so alter the flag */
        if(on)
          (*ts)|=1<<index;    /* set */
        else
          (*ts)&=~(1<<index); /* unset */

        return TRUE;
      }
    }

    ++index;
  }

  return FALSE;
}

/*
 *  void AddWord(PreProcessedDocument *ppd, char *word, tagstate ts);
 *
 *  add the word to word list for document, with weightings for each
 *  word as per ts.
 */

static void AddWord(PreProcessedDocument *ppd, char *wordl, tagstate ts)
{
  int wordcrc=0;
  int newbranchroot=0;
  branchdir newbranchdir=BranchUnknown;
  int iterations=0;
  char *ptr;
  int ucasefield=0;
  char word[32];

  #ifdef DEBUG
  printf("AddWord(PreProcessedDocument *%x, word=%s, tagstate %i)\n",(int)ppd,wordl,ts);
  #endif

  /* process new word, turn it into all lower case - for each upper
   * case character set a bit in ucase field
   */

  ptr=wordl;
  while(*ptr!='\0' && (ptr-wordl)!=31)
  {
    if(isupper(*ptr))
    {
      word[ptr-wordl]=tolower(*ptr);
      ucasefield|=1 << (ptr-wordl);
    } else
      word[ptr-wordl]=*ptr;

    ++ptr;
  }

  word[ptr-wordl]='\0';

  /* make sure this is a valid search term */

  if(!IsValidTerm(word))
    return; /* not worth keeping */

#ifdef STEMMING

  /* now use Porter's stemming algorithm on the word */
  word[(stem(word,0,strlen(word)-1))+1]='\0';
  //printf("\n%s",word);
#endif

  wordcrc=calccrc(word,strlen(word));

  /* first check any existing list entry, or find where our new entry
   * should link from */

  if(ppd->wordlistnext>0)
  {
    int index=0;
    while(index!=ppd->wordlistnext)
    {
      ++iterations;

      /* initially compare crcs */
      if(wordcrc==ppd->wordlist[index].crc)
      {
        if(!strcmp(ppd->wordlist[index].word,word))
        {
          /* found it, increment occurence count */
          ++ppd->wordlist[index].occurrences;

          /* here we increment the weighting for any particular
           * word according to its tag state
           */
          if((ts & tag_title) || (ts & tag_h1))
      {
            /* we only increment weighting if it hasn't been set
             * before, ie. we don't want weighting going up because
             * a term is repeated
             */
            if(ppd->wordlist[index].weighting  == 0)
        {
              ppd->wordlist[index].weighting = 1;
              ++ppd->weightings;
            }
          }

          //printf("\nppd->wordlist[%i].word=%s ppd->wordlist[%i].occurrences=%i iterations=%i\n",index,ppd-
>wordlist[index].word,index,ppd->wordlist[index].occurrences,iterations);
          return;
        }
      }

      /* follow binary tree, if we reach an end (left/right == 0) then
       * keep a pointer to the new branch root
       */

      if(wordcrc<ppd->wordlist[index].crc)
      {
        if(ppd->wordlist[index].left)
          index=ppd->wordlist[index].left;
        else {
          //newbranchroot=&ppd->wordlist[index].left;
          newbranchroot=index;
          newbranchdir=BranchLeft;
          break;
        }
      } else {
        if (ppd->wordlist[index].right)
          index=ppd->wordlist[index].right;
        else {
          //newbranchroot=&ppd->wordlist[index].right;
          newbranchroot=index;
          newbranchdir=BranchRight;
          break;
        }
      }
    }
  }

  if((ppd->wordlistnext % WORDLIST_GRANULARITY) == 0)
  {
    /* we need a new block of memory */
    if(ppd->wordlist)
    {
      //printf("realloc(%x,%x)\n",ppd->wordlist,(ppd->wordlistnext+WORDLIST_GRANULARITY)*sizeof(WordRecord));
      ppd->wordlist=realloc(ppd->wordlist,
                      (ppd->wordlistnext+WORDLIST_GRANULARITY)*sizeof(WordRecord));
    } else {
      //printf("malloc(%x)\n",sizeof(WordRecord)*WORDLIST_GRANULARITY);
      ppd->wordlist=malloc(sizeof(WordRecord)*WORDLIST_GRANULARITY);
      ppd->wordlistnext=0;
    }
  }

  if(!ppd->wordlist)
  {
    fprintf(stderr,"AddWord: Failed to allocate/extend word list\n");
    return;
  }

  /* we have memory allocated */
  strcpy(ppd->wordlist[ppd->wordlistnext].word,word);
  ppd->wordlist[ppd->wordlistnext].occurrences=1;
  ppd->wordlist[ppd->wordlistnext].weighting=0;
  ppd->wordlist[ppd->wordlistnext].crc=wordcrc;
  ppd->wordlist[ppd->wordlistnext].left=0;
  ppd->wordlist[ppd->wordlistnext].right=0;
  ppd->wordlist[ppd->wordlistnext].ucasefield=ucasefield;
  //printf("\nppd->wordlist[%i].word=%s ppd->wordlist[%i].occurrences=%i iterations=%i\n",ppd->wordlistnext,ppd-
>wordlist[ppd->wordlistnext].word,ppd->wordlistnext,ppd->wordlist[ppd->wordlistnext].occurrences,iterations);

  /* here we increment the weighting for any particular word
   * according to its tag state
   */
  if((ts & tag_title) || (ts & tag_h1))
  {
    ppd->wordlist[ppd->wordlistnext].weighting = 1;
    ++ppd->weightings;
  }

  /* add the new branch in to the tree */
  if(newbranchdir!=BranchUnknown) {
    if(newbranchdir==BranchLeft)
      ppd->wordlist[newbranchroot].left=ppd->wordlistnext;
    if(newbranchdir==BranchRight)
      ppd->wordlist[newbranchroot].right=ppd->wordlistnext;
  }

  ++ppd->wordlistnext;
}

/*
 *  void AddWords(PreProcessedDocument *ppd, char *start, char *end
 *                tagstate ts);
 *
 *  add the words between start and end to word list for document,
 *  with weightings for each word as per ts.
 */

static void AddWords(PreProcessedDocument *ppd, char *starto, char *endo,
                     tagstate ts)
{
  char word[32];
  char *ptr, *tmp, *start, *end;
  int wordindex=0,len=0;

  /* first we need to get rid of any entities that may around, ie.
   * &amp; etc. - so as our de-entitise function alters the data in
   * memory we take a copy of what we have, than deentitise it.
   */

  tmp=(char *)malloc((endo-starto)+1);
  if(!tmp)
  {
    fprintf(stderr,"AddWords: Failed to allocate %i bytes for temporary"
                   "store\n",(endo-starto)+1);
    return;
  }

  memcpy(tmp,starto,endo-starto);

  len=EntitiesDeEntitise(tmp,endo-starto);
  start=tmp;
  end=start+len;       /* we have the length, so find end ptr from it */

  /* now check tag state - if it's body text we need to check that our
   * abstract is filled - if not copy as much as we can or as much as
   * we need. ditto titles. Note that this is all before anything
   * detrimental (loss of caps etc) happens to the text.
   */
  if(ts & tag_body)
  {
    if(!ppd->trueabstract) /* if we haven't extracted a true abstract */
    {
      if(ppd->abstractoffset < ABSLEN)
      {
        /* more abstract space left */
        if((ABSLEN - ppd->abstractoffset) > (end-start))
        {
          /* we have more space to fill than we have abstract, copy
           * (end-start)'s worth */
          memcpy(ppd->abstract+ppd->abstractoffset,start,(end-start));
          ppd->abstractoffset+=(end-start);
          ppd->abstract[ppd->abstractoffset]='\0';
        } else {
          /* we have less or equal space to fill than we have abstract, just
           * copy (ABSLEN-ppd->abstractoffset)'s worth */
          memcpy(ppd->abstract+ppd->abstractoffset,start,
                 ABSLEN-ppd->abstractoffset);
          ppd->abstractoffset+=ABSLEN-ppd->abstractoffset;
          ppd->abstract[ppd->abstractoffset-1]='\0';
        }

        /* calls function to replace newlines with spaces, and then
         * remove any double spaces to make optimal use of area
         * available for abstract
         */
        ppd->abstractoffset=makeoneline(ppd->abstract);
        ppd->abstractoffset=striplspace(ppd->abstract);
      }
    }
  }

  if(ts & tag_title) /* we're in a title tag, try to grab title text */
  {
    if(ppd->titleoffset < TITLELEN)
    {
      /* more title space left */
      if((TITLELEN - ppd->titleoffset) > (end-start))
      {
        /* we have more space to fill than we have title, copy
         * (end-start)'s worth */
        memcpy(ppd->title+ppd->titleoffset,start,(end-start));
        ppd->titleoffset+=(end-start);
        ppd->title[ppd->titleoffset]='\0';
      } else {
        /* we have less or equal space to fill than we have title, just
         * copy TITLELEN's worth */
        memcpy(ppd->title+ppd->titleoffset,start,TITLELEN-ppd->titleoffset);
        ppd->titleoffset+=TITLELEN-ppd->titleoffset;
        ppd->title[ppd->titleoffset-1]='\0';
      }
    }
  }

  /* as you were... */

  ptr=start;

  do
  {
    if(isspace(*ptr) || *ptr=='\\' || *ptr=='/' || *ptr=='-' || ptr==end)
    {
      if(wordindex)
      {
        char *newword;

        newword=strippunct(word);
        if(strlen(newword) > 1)
        {
          AddWord(ppd,newword,ts);
        }

        wordindex=0;
      }
    } else {
      if(wordindex<31)
      {
        word[wordindex]=*ptr;
        ++wordindex;
        word[wordindex]='\0';
      }
    }
    ++ptr;
  } while(ptr!=(end+1));

  free(tmp);
}

/* this function is used by the qsort function to determine which of
 * two objects should be treated as greater
 */

int WordRecordSortCompare(const void *a, const void *b)
{
  WordRecord *aa, *bb;

  aa=(WordRecord *)a;
  bb=(WordRecord *)b;

  /* this is where we use our weightings! */
  return((bb->occurrences * (bb->weighting+1)) -
         (aa->occurrences * (aa->weighting+1)));
}
```

```
/*
 *  PreProcessedDocument *PreProcessHTMLDocument(char *file)
 *
 *  Loads a HTML document, returning a PreProcessed document containing
 *  text and the relevant weighting data derived from HTML elements
 *
 */

PreProcessedDocument *PreProcessHTMLDocument(char *data,int length,char *url)
{
  char *dp;                        /* document pointer */
  char *end;                       /* pointer to last byte in document */
  char *tbend;                     /* text block end */
  tagstate ts=0;                   /* current stream state */
  PreProcessedDocument *ppd=NULL; /* preprocessed document */
  MetaElement *metalist=NULL;     /* list of meta tags */

  dp=data;
  end=dp+length;

  /* quickly rattle through the data looking for spurious control
   * characters - if they are present document isn't valid HTML
   */
  while(dp!=end)
  {
    if(iscntrl(*dp) && *dp!=10 && *dp!=9 && *dp!=13)
    {
      fprintf(stderr,"PreProcessHTMLDocument: Document is corrupted or bad HTML containing char %i! Ignoring
%s\n",*dp,url);
      return NULL;
    } else {
      if(*dp==9 || *dp==13)
        *dp=' ';
    }
    ++dp;
  }

  dp=data;

  ppd=(PreProcessedDocument *)malloc(sizeof(PreProcessedDocument));

  if(!ppd)
  {
    fprintf(stderr,"PreProcessHTMLDocument: Failed to allocate %i bytes for PreProcessedDocument
structure\n",sizeof(PreProcessedDocument));
    return NULL;
  }

  /* set up newly allocated structure */
  ppd->wordlist=NULL;
  ppd->wordlistnext=0;
  ppd->follow=1;
  ppd->index=1;
  ppd->trueabstract=0;
  ppd->abstractoffset=0;
  ppd->titleoffset=0;
  ppd->doclength=length;
  ppd->weightings=0;
  time(&ppd->time);
  strcpy(ppd->title,"");
  strcpy(ppd->abstract,"");

  ppd->url=(char *)malloc(strlen(url)+1);
  if(!ppd->url)
  {
    fprintf(stderr,"PreProcessHTMLDocument: Failed to allocate %i bytes for URL string\n",strlen(url)+1);
    return NULL;
  }

  strcpy(ppd->url,url);

  while(dp!=end)
  {
#ifdef DEBUG
    printf("dp=0x%x end=0x%x",(int)dp,(int)end);
#endif
    if(*dp=='<') /* tag! */
    {
      char *endoftag,*tagdata;

      endoftag=dp+1;

      while(*endoftag!='>' && endoftag!=end)
        ++endoftag;

      if(endoftag==end && *endoftag!='>')
      {
        /* html stops with an open tag... bad! ignore document */
        fprintf(stderr,"PreProcessHTMLDocument: Document stops with an "
                       "open tag... ignored doc\n");
        free(ppd);
        return NULL;
      }

#ifdef DEBUG
     printf("tagdata=malloc(%i)\n",endoftag-dp);
#endif
     if((tagdata=malloc(endoftag-dp))==NULL)
     {
        /* out of memory, allocate memory for tag contents - most
         * likely that document isn't valid, ignore it
         */
        fprintf(stderr,"PreProcessHTMLDocument: Could not allocate %i "
                       "bytes for tag contents. Document ignored.",
                       endoftag-dp);
        free(ppd);
        return NULL;
      }
#ifdef DEBUG
      printf("tagdata=0x%x\n",(int)tagdata);
      printf("RED");
#endif

      strncpy(tagdata,dp+1,endoftag-dp-1); /* copy contents of < > */
      tagdata[endoftag-dp-1]='\0';         /* terminate */

      if(!checktag(&ts,tagdata))           /* process any style tags */
      {
        /* tag was not one checktag recognised, look for metas */
        if(!strincmp(tagdata,"meta ",5))
        {
          TagData *td;
          char *result;

          td=TagParse(tagdata);

          /* check for robots meta - contains noindex or nofollow to
           * turn off indexing of page and/or link following
           */

          result=TagGetAttribute("robots",td);
          if(result)
          {
            strlower(result);
            if(strstr(result,"nofollow"))
              ppd->follow=0;
            if(strstr(result,"noindex"))
              ppd->index=0;
            if(strstr(result,"none") || (ppd->follow==0 && ppd->index==0))
            {
              /* document shouldn't be indexed, or have its links followed,
               * so just forget it
               */

              TagFree(td);
              free(ppd);
              return NULL;
            }
          }

          result=TagGetAttribute("keywords",td);
          if(result)
      {
            /* process keywords */
            printf("keywords: %s\n",result);
          }

          result=TagGetAttribute("summary",td);
          if(result)
      {
            /* process keywords */
            printf("summary: %s\n",result);
          }

          TagFree(td);
        }

#ifdef DEBUG
        printf("GREEN");
#endif

        /* anchor - store link */
        if(!strincmp(tagdata,"a ",2))
        {
          if(ppd->follow)
          {
            TagData *td;
            char *href=NULL;

            //printf("TagParse(%s)\n",tagdata);
            td=TagParse(tagdata);
            //printf("done TagParse\n");

            href=TagGetAttribute("href",td);
            if(href)
            {
             /* if href[0]=='#' this is a href local to this document, don't
              * bother following it
              */

              if(href[0]!='#')
              {
                printf("Link to follow... %s\n",href);
              }
            }

            TagFree(td);
          }
        }

#ifdef DEBUG
        printf("BLUE");
#endif

        /* check for a script tag */
        if(!strincmp(tagdata,"script",6))
        {
          char *endofscript;

          endofscript=endoftag;

          /* script tags contain typically javascript code. skip it */
          do {
            endofscript=strchr(endofscript+1,'<');

            if(!strincmp(endofscript,"</script>",9))
            {
              endoftag=endofscript+8; /* skip script block */
              break;
            }
          } while(endofscript!=NULL);

          if(endofscript==NULL)
          {
            fprintf(stderr,"PreProcessHTMLDocument: <script> tag with no close. Document ignored.");
            free(ppd);
            return NULL;
          }
        }

      }

#ifdef DEBUG
      printf("tag(%s -> %i)\n",tagdata,ts);
      printf("tagdata=0x%x\n");
#endif
      free(tagdata);                       /* free tag data */
      /* we can now skip tag */
      dp=endoftag+1;
      continue;
    }

    /* we're not in a tag, so find the start of the next one or the
     * end of the file, and then pass that block to be processed as
     * text
     */

    tbend=strchr(dp,'<');

    if(!tbend)
    {
      printf("no following tag found\n");
      tbend=end; /* if another tag wasn't found process to EOF */
      if(dp==end)
        break;
    } else {
      //printf("follow : %s\n",tbend);
    }

    //printf("[[");
    if(ppd->index)
    {
#ifdef DEBUG
      printf("AddWords(0x%x,0x%x,0x%x,0x%x)\n",ppd,dp,tbend,ts);
#endif
      AddWords(ppd,dp,tbend,ts);
    }

#ifdef DEBUG
    printf("dp=0x%x tbend=0x%x end=0x%x\n",dp,tbend,end);
#endif
    dp=tbend;
  }

  printf("quicksorting..\n");

  qsort((void *)ppd->wordlist, ppd->wordlistnext, sizeof(WordRecord), WordRecordSortCompare);

  /* This quickly determines how many of the top terms should be submitted
   * to the index - this could be made proportional to the length of the
   * document if needed.
   */
  if(ppd->wordlistnext < MAXTERMS)
    ppd->terms=ppd->wordlistnext;
  else
    ppd->terms=MAXTERMS;

  /* If any of the terms in the list are 1 with no weighting, lop them
   * off the terms to be included in the index
   */
  while(ppd->terms)
  {
    if(ppd->wordlist[ppd->terms].occurrences == 1 &&
       ppd->wordlist[ppd->terms].weighting == 0)
      --ppd->terms;
    else
      break;
  }


  {
    int index=0;

    while(index!=ppd->wordlistnext)
    {
      printf("%4i %2i %32s %8x\n",index, ppd->wordlist[index].occurrences,ppd->wordlist[index].word,ppd-
>wordlist[index].crc);
      ++index;
    }
  }

  return ppd;
}

/* This is the code to submit a document to the database */
int SubmitDocument(PreProcessedDocument *ppd)
{
  PreProcDoc *ppdn;
  int ppdnl; /* size needed for transfer block */
  int i=0, length=0;
  WordEntry *we;
  char *urlp;
  char *snierror=NULL;
  sni_response sr;

  /*** Stage 1 : first we need to build a PreProcDoc structure ready to
   *** send to the server (see preprocdoc.h)
   */
  if(!ppd->url)
  {
    fprintf(stderr,"SubmitDocument: Cannot submit a document with no url in PreProcessedDocument structure\n");
    return 1;
  }

  if(!ppd->terms)
  {
    fprintf(stderr,"SubmitDocument: (non-problem) document %s has no indexable terms\n",ppd->url);
    return 0;
  }

  /* calc length for header structure, weights list and url string */
  length=sizeof(PreProcDoc)+(ppd->terms*sizeof(WordEntry))+strlen(ppd->url)+1;

  ppdn=malloc(length);
  if(!ppdn)
  {
    fprintf(stderr,"SubmitDocument: Cannot allocate %i bytes for PreProcDoc structure!\n",length);
    return 1;
  }

  /* clear memory */
  bzero(ppdn,length);

  /* now build up the PreProcDoc structure */
  strcpy(ppdn->abstract,ppd->abstract);
  strcpy(ppdn->title,ppd->title);
  ppdn->doclength=ppd->doclength;
  ppdn->wordcount=ppd->wordlistnext;
  memcpy(&ppdn->time,&ppd->time,sizeof(time_t));
  ppdn->indexedwords=ppd->terms;

  /* obtain pointer to first WordEntry */
  we=(WordEntry *)(ppdn+1);

  /* copy the terms to index */
  while(i!=ppd->terms)
  {
    we[i].weighting=ppd->wordlist[i].weighting;
    we[i].occurrences=ppd->wordlist[i].occurrences;
    strcpy(we[i].word,ppd->wordlist[i].word);
    ++i;
  }

  /* finally copy the url */
  memcpy((char *)we+(sizeof(WordEntry)*ppd->terms),
         ppd->url,strlen(ppd->url)+1);

#ifdef DEBUG
  debug_dumper(ppdn,length,stdout,DUMP_WORD);
  debug_dump_preprocdoc(ppdn);
#endif

  /*** Stage 2 : Now we submit the URL ***/
  snierror=sni_send(SNISERVER,SNIPORT,SNI_DOCSUBMISSION,ppdn,length,&sr);

  if(snierror)
  {
    log_event("sni_send error : %s", snierror);
    free(ppdn);
    return 1;
  } else {
    if(sr.type==SNI_FAIL)
    {
       log_event("SubmitDocument: SNI_FAIL %s",
                 sr.data ? sr.data : "no text" );
       free(ppdn);
       return 1;
    } else {
      if(sr.type!=SNI_OK)
      {
        log_event("SubmitDocument: Unexpected SNI response, %i - data length %i",sr.type,sr.length);
        return 1;
      }
    }
  }

  /*** Stage 3 : Tidy up ***/
  free(ppdn);
  return 0;
}

/* SIGSEGV signal handler */
void sig_segv()
{
  printf("Parser got confused...\n");
  exit(1);
}

int main(int argc, char **argv)
{
  char *data;
  int length,error;
  double t;
  PreProcessedDocument *ppd;

  setbuf(stdout,NULL);
  setbuf(stderr,NULL);
  printf("preprochtml ("__DATE__" "__TIME__")");

#ifdef DEBUG
  printf(" DEBUG\n");
#else
  printf("\n");
#endif

  if(!argv[1])
  {
    fprintf(stderr,"No file!\n");
    return 0;
  }

  printf("%s processing...\n",argv[1]);

  load_stopwords("/home/thowat/project/stopwords");

  data=load_malloc(argv[1],&length,&error);

  if(error)
  {
    fprintf(stderr,"Failed to load %s\n",argv[1]);
    return 0;
  }

  clock();

  /* signal handler just incase we overrun/underrun a buffer */
  signal(SIGSEGV,(void (*)())sig_segv);

  if(data)
    ppd=PreProcessHTMLDocument(data,length,argv[1]);

  if(!ppd || !data)
    return 0;

  t=(double)clock()/(double)CLOCKS_PER_SEC;

  if(t>0)
    printf("%f seconds to process %i bytes, %f bytes/sec\n",
t,length,(double)length/t);

  printf("Document title    = %s\n",ppd->title);
  printf("Document abstract = %s\n",ppd->abstract);

  if(SubmitDocument(ppd))
    printf("There were significant problems adding %s",argv[1]);

  free_stopwords();

  printf("exiting OK, 0\n");
  return 0;
}
```

F.17 shavtimer.c

```
/* shavtimer.c
 * SHared AVerage Timer
 * Tony Howat 2000
 *
 * Times operations using simple start/stop, and will then
 * return an average operation duration in ms on request.
 * Clever bit is the timers are kept in shared memory, and
 * so this will work for forked() servers.
 *
 */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>
#include "shavtimer.h"
#include "global.h"

#define SHAVSHM_KEY 7900  /* our shared memory key */
#define PERMS       0666

typedef struct
{
  clock_t totalclockcycles;
  int     totaloperations;
  clock_t last;
} SharedAverageTimer;

SharedAverageTimer *shavt;
int shmid=0; /* shared memory id */
int ppid=0;  /* parent process id */

/* these are our local start times */
clock_t starttime[shavt_count]={0};

#ifdef VIRTUALTIME
long millisecondclock=0;
#endif
```

```
/* the exit handler makes sure we drop shared memory */
void shavtimer_exithandler(void)
{
  if(shmid)
  {
    if(ppid==getpid())
    {
      log_event("Parent dropping shared memory.");
      shavtimer_end();
    } else {
      log_event("Child dropping shared memory.");
      shavtimer_child_end();
    }
  }
}

/* initialise shared timers -- called in parent process
 * returns -1 on failure
 */
int shavtimer_init()
{
  /* keep a copy of parent process id so we know what to do on atexit */
  ppid=getpid();

  /* create the shared memory segment */
  if((shmid = shmget(SHAVSHM_KEY,
                     sizeof(SharedAverageTimer)*(int)shavt_count,
                     PERMS | IPC_CREAT)) < 0 )
  {
    log_event("shavtimer_init: Failed to create shared memory segment, attempting to open and clear any existing
shared memory block");

    /* create the shared memory segment */
    if((shmid = shmget(SHAVSHM_KEY,
                       sizeof(SharedAverageTimer)*(int)shavt_count,
                       0)) < 0 )
    {
      log_event("shavtimer_init: Failed to find any previously allocated shared memory");
      return -1;
    }
  }

  /* register our exit handler */
  atexit(shavtimer_exithandler);

  /* attach it */
  if((shavt = (SharedAverageTimer *)shmat(shmid, (char *)0, 0))
     == (SharedAverageTimer *)-1)
  {
    log_event("shavtimer_init: Failed to attach shared memory");
    return -1;
  }

  memset(shavt,'\0',sizeof(SharedAverageTimer)*(int)shavt_count);

  return 0;
}

#ifdef VIRTUALTIME

/* SIGPROF signal handler */
void sig_profalarm()
{
  struct itimerval itv,oval;

  millisecondclock+=10;
  itv.it_interval.tv_sec=0;
  itv.it_interval.tv_usec=1000;
  itv.it_value.tv_sec=0;
  itv.it_value.tv_usec=1000;
  setitimer(ITIMER_PROF,&itv,&oval);
}

#endif

/* initialise shared timers -- called in child process
 * returns -1 on failure
 */
int shavtimer_child_init()
{
#ifdef VIRTUALTIME
  struct itimerval itv,oval;
#endif

  /* create the shared memory segment */
  if((shmid = shmget(SHAVSHM_KEY,
                     sizeof(SharedAverageTimer)*(int)shavt_count,
                     0)) < 0 )
  {
    fprintf(stderr,"shavtimer_child_init: Failed to claim shared memory\n");
    return -1;
  }

  /* attach it */
  if((shavt = (SharedAverageTimer *)shmat(shmid, (char *)0, 0))
             == (SharedAverageTimer *)-1)
  {
    fprintf(stderr,"shavtimer_child_init: Failed to attach shared memory\n");
    return -1;
  }

#ifdef VIRTUALTIME
  /* we need to set up an itimer on our process, so we can count
   * virtual time - register our signal handler and set timer
   */
  millisecondclock=0;
  signal(SIGPROF,(void (*)())sig_profalarm);

  itv.it_interval.tv_sec=0;
  itv.it_interval.tv_usec=10000; /* 10ms */
  itv.it_value.tv_sec=0;
  itv.it_value.tv_usec=10000; /* 10ms */
  setitimer(ITIMER_PROF,&itv,&oval);
#endif

  return 0;
}

/* close shared timers -- called in child process, returns
 * -1 on failure
 */
int shavtimer_child_end()
{
  if(shmid)
  {
    /* detach it */
    if(shmdt(shavt) <0)
    {
      fprintf(stderr,"shavtimer_child_end: Failed to detach shared memory\n");
      return -1;
    }
  }

  shmid=0;

  return 0;
}

/* close shared timers -- called in parent process, returns
 * -1 on failure
 */
int shavtimer_end()
{
  if(shmid)
  {
    /* detach it */
    if(shmdt(shavt) < 0)
    {
      fprintf(stderr,"shavtimer_end: Failed to detach shared memory\n");
      return -1;
    }

    /* remove it */
    if(shmctl(shmid,IPC_RMID, (struct shmid_ds *) 0) < 0)
    {
      fprintf(stderr,"shavtimer_end: Failed to remove shared memory\n");
      return -1;
    }
  }

  shmid=0;

  return 0;
}

/* time a new operation
 * returns -1 if an operation is already being timed
 */
int shavtimer_starttimer(shavt_timer st)
{
  if(starttime[st])
  {
    log_event("Couldn't start timer %i",st);
    return -1;
  }

#ifdef VIRTUALTIME
  starttime[st]=millisecondclock;
#else
  starttime[st]=clock();
#endif
  return 0;
}

/* stop timing an operation
 * returns -1 if no operation was being timed
 */
int shavtimer_stoptimer(shavt_timer st)
{
  int elapsed;

#ifdef VIRTUALTIME
  elapsed=millisecondclock-starttime[st];
  shavt[st].last=elapsed;
#else
  elapsed=clock()-starttime[st];
  shavt[st].last=(int)((double)elapsed/(double)(CLOCKS_PER_SEC/1000000));
#endif
  shavt[st].totalclockcycles+=elapsed;
  shavt[st].totaloperations++;
  starttime[st]=0;

  return 0;
}

/* stop timing an operation, and don't include in average
 * returns -1 if no operation was being timed
 */
int shavtimer_droptimer(shavt_timer st)
{
  if(!starttime[st])
    return -1;
  starttime[st]=0;

  return 0;
}

/* return the number of milliseconds taken on average per
 * operation
 */
#ifdef VIRTUALTIME
int shavtimer_getaverage(shavt_timer st)
{
  clock_t averagecycles;

  if(shavt[st].totaloperations == 0)
    return 0;

  averagecycles=shavt[st].totalclockcycles/(clock_t)shavt[st].totaloperations;
  return((double)averagecycles/(double)(CLOCKS_PER_SEC/1000000));
}
#else
int shavtimer_getaverage(shavt_timer st)
{
  int average;

  if(shavt[st].totaloperations == 0)
    return 0;

  average=shavt[st].totalclockcycles/shavt[st].totaloperations;
  return(average);
}
#endif

/* return the number of operations timed */
int shavtimer_getoperations(shavt_timer st)
{
  return shavt[st].totaloperations;
}

/* return last measured time */
int shavtimer_getlast(shavt_timer st)
{
  return shavt[st].last;
}

/* reset a timer */
void shavtimer_reset(shavt_timer st)
{
  shavt[st].totaloperations=0;
  shavt[st].totalclockcycles=0;
}
```

F.18 shavtimer.h

```
/* shavtimer.h
 * SHared AVerage Timer
 * Tony Howat 2000
 *
 * Times operations using simple start/stop, and will then
 * return an average operation duration in ms on request.
 * Clever bit is the timers are kept in shared memory, and
 * so this will work for forked() servers.
 *
 */

#include <time.h>

typedef enum {
  shavt_retrieve=0,
  shavt_search,
  shavt_submit,
  shavt_count
} shavt_timer;

/* initialise shared timers -- called in parent process
 * returns -1 on failure
 */
int shavtimer_init();

/* initialise shared timers -- called in child process
 * returns -1 on failure
 */
int shavtimer_child_init();

/* close shared timers -- called in child process, returns
 * -1 on failure
 */
int shavtimer_child_end();

/* close shared timers -- called in parent process, returns
 * -1 on failure
 */
int shavtimer_end();

/* time a new operation
 * returns -1 if an operation is already being timed
 */
int shavtimer_starttimer(shavt_timer st);

/* stop timing an operation
 * returns -1 if no operation was being timed
 */
int shavtimer_stoptimer(shavt_timer st);

/* stop timing an operation, don't include in average
 * returns -1 if no operation was being timed
 */
int shavtimer_droptimer(shavt_timer st);

/* return the number of milliseconds taken on average per
 * operation
 */
int shavtimer_getaverage(shavt_timer st);

/* return the number of operations timed */
int shavtimer_getoperations(shavt_timer st);

/* return the duration of last operation */
int shavtimer_getlast(shavt_timer st);

/* reset a timer */
void shavtimer_reset(shavt_timer st);
```

F.19 sni.h

```
/* sni.h
 *
 * Simple Network Interface (Shared definitions)
 *
 */

#ifndef __SNI_H
#define __SNI_H 1

typedef struct _sni_response {
  int length;
  char *data;
  int type;
} sni_response;

#endif

#define SNISERVER "mipc-21"
#define SNIPORT   1234

/* message types */
#define SNI_MSGTEST         1    /* debugging test */
#define SNI_DOCSUBMISSION   100  /* submit a document to db */
#define SNI_GETDOCRECORD    101  /* request a DocumentRecord
                                  * structure - pass a hash of
                                  * base form URL, and it will
                                  * return DocumentRecord followed
                                  * by URL string as SNI_DOCRECORD */
#define SNI_GETDOCCHUNK     102  /* same as above but query by
                                  * DocRecord chunk_id (faster) */
#define SNI_QUERY           103  /* start a query - pass a int number
                                  * of terms followed by that number
                                  * of fixed length query terms
                                  * (TERMSTRINGLENGTH) each
                                  * server returns a SNI_QUERYRESULTS
                                  */
#define SNI_MSGINFO         104  /* request server stats, returned
                                  * as SNI_OK msg or SNI_FAIL */

/* response types */
#define SNI_OK              200  /* generic OK */
#define SNI_FAIL            201  /* generic failure - any
                 * data is char indicating
                                  * problem */
#define SNI_DOCRECORD       202  /* DocumentRecord followed by URL
                                  * string */
#define SNI_QUERYRESULTS    203  /* Query results : a first integer
                                  * giving the number of matches,
                                  * followed by pairs of document
                                  * hashes and relevance weightings
                                  * - sorted best first
                                  */
```

F.20 snic.c

```
/* snic.c
 *
 * Simple Network Interface (Client)
 *
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "snic.h"
```

```
#define socketclose(x) close(x)

/* sni_send
 *
 * Send an arbitary block of data to a remote system using a TCP stream,
 * recieve any response (either a response code or a whole block of data)
 *
 * char *dest        destination hostname
 * int  port         IP port number
 * int  type         transfer type
 * void *data        data to send
 * int length        length of data to send
 * sni_response *sr  pointer to sni_response structure to be filled
 *                   with returned data and response code
 *
 * returns : char * containing a simple error message or NULL
 *
 */

char *sni_send(char *dest, int port, int type, void *data, int length,
               sni_response *sr)
{
  struct hostent *host;
  int sd;
  int one = 1;
  struct sockaddr_in address;
  char hostname[128];
  unsigned char buffer[1024];
  char *rdata=NULL;
  int resplength=0, size=0, respcode=0;

  strcpy(hostname,dest);

  sr->length=0;
  sr->data=0;
  sr->type=0;

  bzero((char *)&address,sizeof(address));
  address.sin_port=htons(port);
  address.sin_family=AF_INET; /* fill in address family */

  /* is hostname a dotted address? */
  if((address.sin_addr.s_addr=inet_addr(dest)) == (u_long)-1)
  {
    if((host=gethostbyname(hostname)) == NULL)
      return("Invalid host name");

    strcpy(hostname,host->h_name);

    /* fill in address */
    address.sin_addr.s_addr=*(u_long *)host->h_addr;
  } else {
    if((host=gethostbyaddr((char *)&address.sin_addr, 4, AF_INET)) != NULL)
      strcpy(hostname,host->h_name);
  }

  if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP)) <0)
    return("Socket creation failed");

  /* if(ioctl(sd,FIONBIO, &one) < 0)
     return("Failed to set socket non-blocking"); */

  /* try the connection */

  if(connect(sd,(struct sockaddr *)&(address), sizeof(address)) <0)
  {
    if(errno!=EINPROGRESS)
    {
      shutdown(sd,2);
      socketclose(sd);
      return("Connect failed");
    }
  }

  /* now send the length, type and data */
  if((send(sd,(char *)&type,4,0)) != sizeof(int))
    return("Failed to send type");
  if((send(sd,(char *)&length,4,0)) != sizeof(int))
    return("Failed to send length");
  if((send(sd,(char *)data,length,0)) != length)
    return("Failed to send data");

  /* now wait for a response */
  size=recv(sd,(char *)&respcode,4,0);
  /* now wait for a resplength */
  size=recv(sd,(char *)&resplength,4,0);

  if(resplength)
  {
    int gotsofar=0;

    /* get some data */
    size=recv(sd,(char *)buffer,1024,0);

    if(size<1 && errno!=EWOULDBLOCK)
    {
      close(sd);
      if(rdata)
        free(rdata);
      return("Connection dropped by client session");
    }

    while(size > 0)
    {
      if(rdata)
        rdata=(void *)realloc(rdata,gotsofar+size+1);
      else
        rdata=(void *)malloc(size+1);

      if(!rdata)
      {
        fprintf(stderr,"sni_send: Failed to allocate memory for incoming rdata (%i bytes, message
%i)\n",size+1,type);
        close(sd);
        return("No memory for rdata");
      }

      memcpy((char *)rdata+gotsofar,(char *)buffer,size);
      gotsofar=gotsofar+size;

      if(resplength < gotsofar)
      {
        close(sd);
        free(rdata);
        return("Server tried to send more data than it promised");
      }

      #ifdef DEBUG
      fprintf(stderr,"Got %i of %i (type %x)\n",gotsofar,length,type);
      #endif

      if(resplength == gotsofar)
      {
        /* we're all done here */
        sr->length=resplength;
        sr->data=rdata;
        sr->type=respcode;
        close(sd);
        return 0;
      }

      size=recv(sd,(char *)buffer,1024,0);
    }
  } else {
    /* we're all done here */
    sr->length=resplength;
    sr->data=rdata;
    sr->type=respcode;
    close(sd);
    return 0;
  }

  return 0;
}

#ifdef SNICTEST

int main(void)
{
  char test[]="This is a test\n";
  char *err;
  sni_response sr;

  err=sni_send("mipc-21",1234,1,test,strlen(test)+1,&sr);
  if(err)
    printf("%s\n",err);

  printf("sr.type=%i sr.length=%i sr.data=0x%x\n",sr.type,sr.length,sr.data);
  if(sr.data!=0)
  {
    sr.data[sr.length]='\0';
    printf("%s\n",sr.data);
  }

  return(0);
}

#endif
```

F.21 snic.h

```
/* snic.h
 *
 * Simple Network Interface (Client)
 *
 */

#include "sni.h"

#ifndef __SNIC_H
#define __SNIC_H 1

/* sni_send
 *
 * Send an arbitary block of data to a remote system using a TCP stream,
 * recieve any response (either a response code or a whole block of data)
 *
 * char *dest        destination hostname
 * int  port         IP port number
 * int  type         transfer type
 * void *data        data to send
 * int length        length of data to send
 * sni_response *sr  pointer to sni_response structure to be filled
 *                   with returned data and response code
 *
 * returns : char * containing a simple error or NULL
 *
 */

char *sni_send(char *dest, int port, int type, void *data, int length,
               sni_response *sr);

#endif
```

F.22 stopwords.c

```
/*
 * stopwords.c
 * Tony Howat, 1999/2000
 *
 * Stores a dictionary of stopwords which should not be searched for
 * or included in indexes
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "util.h"

/* list of stop words separated by commas or whitespace, just loaded as
 * a plain block of text (stopwords list is short, no point in using
 * quicker methods at the moment
 */
static char *stopwords=NULL;
static int stopwordslength=0;

/* returns true if *word is a valid search term */
int IsValidTerm(char *word)
{
  char *ptr;

  ptr=word;

  if(strlen(word)<2)
   return FALSE;

  /* stage 1 - check it contains at least some letters */

  while(*ptr!=0)
  {
    if(isupper(*ptr) || islower(*ptr))
      break; /* search term might be ok */
    ++ptr;
  }

  if(*ptr=='\0')
    return FALSE; /* bad search term, contains no alpha chars */

  /* stage 2 - check against stopwords */
  return(!stopwords_search(word));
}


/*
 *  void stopwords_load(char *file)
 *
 *  Loads a list of stopwords
 *
 */

void load_stopwords(char *file)
{
   int error;

   stopwords=load_malloc(file,&stopwordslength,&error);

   switch(error)
   {
     case uOPENFAIL:
       fprintf(stderr,"No stopwords file (%s)\n",file);
       break;
     case uNOMEMORY:
       fprintf(stderr,"No memory for stopwords file! Quitting.\n");
       exit(1);
       break;
   }
}
/*
 *  void stopwords_free(void)
 *
 *  Loses stopwords list
 *
 */

void free_stopwords(void)
{
   free(stopwords);
}

/*
 *  int stopwords_search(char *word)
 *
 *  Searches stopwords list for a word, returns true if it is a stopword
 *
 */

int stopwords_search(char *word)
{
   char *stopwordindex=NULL;

   if(!stopwords)
     return FALSE;

#ifdef DEBUG
   printf("stopwords_search(%s)\n",word);
#endif

   stopwordindex=stopwords;

   while(strstr(stopwordindex,word) &&
         stopwordindex<(stopwordindex+stopwordslength))
   {
     stopwordindex=strstr(stopwordindex,word);

     if(*(stopwordindex+strlen(word))!='\0' &&
        *(stopwordindex+strlen(word))!=',' &&
        !isspace(*(stopwordindex+strlen(word))))
     {
        /* character following is not whitespace, comma or null, so no match */
        stopwordindex+=strlen(word);
        continue;
     }

     if(stopwordindex!=stopwords)
     {
       /* Check previous character - if whitespace or comma it's a match */
       if(*(stopwordindex-1)==',' || *(stopwordindex-1)=='\n')
         return TRUE;
       else {
        stopwordindex+=strlen(word);
        continue;
       }
     } else {
       return TRUE; /* match starts at start of file */
     }
   }

   return FALSE;
}
```

F.23 stopwords.h

```
/*
 * stopwords.h
 * Tony Howat, 1999/2000
 *
 * Stores a dictionary of stopwords which should not be searched for
 * or included in indexes
 *
 */


/*
 *  void stopwords_load(char *file)
 *
 *  Loads a list of stopwords
 *
 */

void load_stopwords(char *file);

/*
 *  void stopwords_free(void)
 *
 *  Loses stopwords list
 *
 */

void free_stopwords(void);

/*
 *  int stopwords_search(char *word)
 *
 *  Searches stopwords list for a word, returns true if it is a stopword
 *
 */

int stopwords_search(char *word);


/* returns true if *word is a valid search term */
int IsValidTerm(char *word);
```

F.24 tags.c

```
/*
 * tags.c
 * Tony Howat, 1999/2000
 *
 * Parses HTML tags into a more useful form (structure and linked
 * list of attributes), and provides basic search operations.
 * More complex operations may be done externally by using the
 * structures directly.
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "tags.h"

/*
 *  TagData *TagParse(char *tagdata);
 *
 *  Pass a tag and its parameters with no <>. This routine will turn
 *  the tag into a useful TagData/TagAttribute structure ready to
 *  be searched using TagGetAttribute.
 *
 */

TagData *TagParse(char *tagdata)
{
  TagData *td=NULL;
  TagAttribute *ta=NULL;
  char *ptr;
  char *tagdatastart;

  return NULL;

  tagdatastart=tagdata;

  /* allocate memory for TagData */

  td=malloc(sizeof(TagData));
  if(!td)
  {
    fprintf(stderr,"TagParse: Failed to allocate %i bytes for TagData\n",
            sizeof(TagData));
    return NULL;
  }

  td->name=NULL;
  td->attributes=NULL;

  /* [TAG name="foo" content="baa" scheme="moo" http-equiv="woo"]
   *  ^^^ pull this out
   */

  ptr=tagdata;
  while(!isspace(*ptr) && *ptr!=0)
    ++ptr; /* look for first space */
  while(isspace(*ptr))
    ++ptr; /* skip any further spaces */

  /* if no attributes ptr will point to end of tagdata */

  td->name=malloc((ptr-tagdata)+1);
  if(!td->name)
  {
    fprintf(stderr,"TagParse: Failed to allocate %i bytes for TagData->name\n",
            ptr-tagdata);
    free(td);
    return NULL;
  }

  while(tagdata!=ptr)
  {
    td->name[tagdata-tagdatastart]=*tagdata;
    ++tagdata;
  }
  td->name[tagdata-tagdatastart]='\0';

  /* Now pull out each name and attribute value in turn
   * [name="foo" content="baa" scheme="moo" http-equiv="woo"] */

  while(*tagdata!='\0')
  {
    //char attname[64]="";
    //int attindex=0;
    char waitfor='\"';
    char *paramdata=NULL, *paramdataend=NULL, *paramdatastart=NULL;
    char *attrdata=NULL, *attrdataend=NULL, *attrdatastart=NULL;

    while(isspace(*tagdata)) /* skip whitespace */
      ++tagdata;

    if(!isalpha(*tagdata))
    {
      fprintf(stderr,"TagParse: Bad <%s>, ignoring\n",tagdatastart);
      return NULL;
    }

    /* get attribute name */

    attrdataend=strchr(tagdata,'=');
    attrdata=malloc((strchr(tagdata,'=')-tagdata)+1);
    attrdatastart=attrdata;

    if(!attrdata)
    {
      fprintf(stderr,"TagParse: Failed to allocate memory for <%s>, ignoring\n",
              tagdatastart);
      return NULL;
    }
    while(tagdata!=attrdataend)
    {
      *attrdata=*tagdata;
      ++tagdata; ++attrdata;
    }

    *attrdata='\0';

    /* done attribute name */

    if(*tagdata!='=')
    {
      fprintf(stderr,"TagParse: Bad <%s>, ignoring\n",tagdatastart);
      return NULL;
    }

    ++tagdata;

    while(*tagdata!='\'' && *tagdata!='\"') /* skip to start of value */
      ++tagdata;

    waitfor=*tagdata;
    ++tagdata;

    paramdataend=strchr(tagdata,waitfor);
    paramdata=malloc((strchr(tagdata,waitfor)-tagdata)+1);
    paramdatastart=paramdata;

    if(!paramdata)
    {
      fprintf(stderr,"TagParse: Failed to allocate memory for <%s>, ignoring\n",
              tagdatastart);
      return NULL;
    }

    while(tagdata!=paramdataend)
    {
      *paramdata=*tagdata;
      ++tagdata; ++paramdata;
    }

    *paramdata='\0';

#ifdef TAGDEBUG
    printf("\n%s = %s\n",attrdatastart,paramdatastart);

    printf("{%s}",tagdata);
#endif

    /* fill it in */

    ta=malloc(sizeof(TagAttribute));
    if(!ta)
    {
      fprintf(stderr,"TagParse: Failed to allocate %i bytes for TagAttribute\n",
              sizeof(TagAttribute));
      return NULL;
    }

    ta->name=attrdatastart;
    ta->content=paramdatastart;
    ta->next=NULL;

    /* add to linked list */

    if(td->attributes==NULL)
    {
      td->attributes=ta;
    } else {
      TagAttribute *nta;

      nta=td->attributes;
      while(nta->next!=NULL)
        nta=nta->next;

      nta->next=ta;
    }

    tagdata=paramdataend+1;
  }

  return td;
}

/*
 *  char *TagGetAttribute(char *attribute,TagData *td);
 *
 *  td is a structure previously created by ParseTag. attribute
 *  is the name of the tag attribute to retrieve.
 *
 */

char *TagGetAttribute(char *attribute,TagData *td)
{
  TagAttribute *ta=NULL;

  if(td==NULL)
   return NULL;

  ta=td->attributes;

  while(ta!=NULL)
  {
    if(!stricmp(ta->name,attribute))
      return ta->content;

    ta=ta->next;
  }

  return NULL;
}

/*
 *  void TagFree(TagData *td);
 *
 *  Frees structures created by TagParse
 *
 */

void TagFree(TagData *td)
{
  TagAttribute *ta=NULL;

  if(!td)
    return;

  ta=td->attributes;

  /* free TagAttributes linked list */

  while(ta!=NULL)
  {
    TagAttribute *tofree;
#ifdef TAGDEBUG
    printf("freeing %s\n",ta->name);
#endif
    free(ta->name);
#ifdef TAGDEBUG
    printf("freeing %s\n",ta->content);
#endif
    free(ta->content);
    tofree=ta;
    ta=ta->next;
#ifdef TAGDEBUG
    printf("freeing %x (tofree)\n",(int)tofree); free(tofree);
    printf("ta now = %x\n",(int)ta);
#endif
  }

  /* finally free tagdata */
#ifdef TAGDEBUG
  printf("freeing %x (td)\n",(int)td);
#endif
  free(td);
}
```

F.25 tags.h

```
/*
 * tags.h
 * Tony Howat 1999/2000
 *
 * Parses HTML tags into a more useful form (structure and linked
 * list of attributes), and provides basic search operations.
 * More complex operations may be done externally by using the
 * structures directly.
 *
 */

typedef struct _TagAttribute
{
  char *name;        /* an argument in a HTML meta tag */
  char *content;
  struct _TagAttribute *next;
} TagAttribute;

typedef struct
{
  char *name;         /* contents of a HTML meta tag */
  TagAttribute *attributes;
} TagData;

/*
 *  TagData *TagParse(char *tagdata);
 *
 *  Pass a tag and its parameters with no <>. This routine will turn
 *  the tag into a useful TagData/TagAttribute structure ready to
 *  be searched using TagGetAttribute.
 *
 */

TagData *TagParse(char *tagdata);

/*
 *  char *TagGetAttribute(char *attribute,TagData *td);
 *
 *  td is a structure previously created by ParseTag. attribute
 *  is the name of the tag attribute to retrieve.
 *
 */

char *TagGetAttribute(char *attribute,TagData *td);

/*
 *  void TagFree(TagData *td);
 *
 *  Frees structures created by TagParse
 *
 */
void TagFree(TagData *td);
```

F.26 uri.c

```
/*
 * uri.c
 * Tony Howat, 1999/2000
 *
 * Routines for munging URIs, dealing with relative URIs etc.
 * ** tested and working, but needs debug info stripping **
 *
 * References RFC1808
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "util.h"
#include "uri.h"

#define RESOLVENAMES 1

#ifdef RISCOS
#include "arpa/inet.h"
#include "netdb.h"
#include "netinet/in.h"
#include "sys/byteorder.h"
#include "sys/errno.h"
#include "sys/ioctl.h"
#include "sys/socket.h"
#else
#include "sys/types.h"
#include "sys/socket.h"
#include "sys/ioctl.h"
#include "netdb.h"
#include "resolv.h"
#endif

static void URIPack(URIExtruded *URI);

/* hex to decimal table */
static const char dvalt[]="0123456789ABCEF";

/* returns decimal value of a hex character, or -1 if not hex */
int dval(char c)
{
  register int i=0;

  c=toupper(c);

  while((dvalt[i]!='\0') && (dvalt[i]!=c))
    ++i;

  if(dvalt[i])
    return i;
  else
    return -1;
}

size_t hex_escape_path_cpy(char *dest, const char *src)
{
  size_t count = 0;

  for (;;) {
    const char p = *src++;

    if (!p) {
      *dest = p;
      return count;
    }

    /* see if this character should be escaped */
    if ((p==0x7f) || (p<=' '))
    {
      const register unsigned int ch = (unsigned int) p;

      *dest++ = '%';
      *dest++ = dvalt[ch / 16];
      *dest++ = dvalt[ch % 16];
      count += 3;
    } else {
      *dest++ = p;
      ++count;
    }
  }
}

static char *URIExtrudePath(URIExtruded *ue, char *path)
{
  char *URIew;
  char *URIewl;
  int index=0;

  URIew=path;
  URIewl=URIew+strlen(URIew);

  printf("URIewl = %s ... *(URIewl-1) = %c\n",URIewl, *(URIewl-1));
```

```
/* if the path ends with / it's a directory, otherwise it's a
 * file, so we need to pull off a leafname
 */

if(*(URIewl-1)!='/')
{
  char *lnstart, *wkptr;

  lnstart=URIewl-1;

  printf("leafname\n");

  /* find the start of the leafname */

  while(lnstart>path)
  {
    printf("[%c]",*lnstart);
    if(*lnstart=='/')
      break;
    --lnstart;
  }

  if(*lnstart!='/')
    lnstart=path; /* leafname must be all of path part of URL */

  /* make sure that the leafname isn't .. or . -
   * in which case it isn't a leafname, it's a part of the path
   * description. uhm.
   */

  if((strcmp(lnstart,"..")) && (strcmp(lnstart,".")))
  {
    /* if we're adding on a relative URL there may be an
     * existing leafname component. If so, free it.
     */

    if(ue->leafname)
      free(ue->leafname);

    /* now alloc memory for leafname and copy */
    ue->leafname=malloc((URIewl-lnstart)+2);
    if(!ue->leafname)
    {
      fprintf(stderr,
              "URIExtrudePath: Failed to malloc %i bytes for"
              "ue->leafname\n",
              (URIewl-lnstart)+2);
      return NULL;
    }

    wkptr=(lnstart+1); /* points to first char of leafname */
    index=0;

    printf("wkptr=%x URIewl=%x\n",wkptr,URIewl);
    while(wkptr<URIewl)
    {
      ue->leafname[index]=*wkptr;
      ++wkptr;
      ++index;
    }
    ue->leafname[index]='\0';

    /* now make the end of the path correct */
    URIewl=lnstart;

    printf("ue->leafname=%s\n",ue->leafname);
  } else {
    printf("norra leafname\n");
    URIewl++;
  }
}

/*
 * This code extracts the path into a linked list of URIPathElements
 *
 * http://user@www.moo.org:8080/directory1/directory2/index.html
 *                              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
 */

while(URIew<URIewl)
{
  char *elend;
  URIPathElement *upe;
  int cpl=0;

  printf("URIew=%s\n",URIew);

  elend=strchr(URIew,'/');
  if(!elend)
  {
    elend=URIewl-1;
  }

  /* allocate memory for linked list entry */

  upe=malloc(sizeof(URIPathElement));
  if(!upe)
  {
    fprintf(stderr,
            "URIExtrude: Failed to malloc %i bytes for URIPathElement\n",
            sizeof(URIPathElement));
    return NULL;
  }

  upe->next=NULL;
```

```
    /* allocate memory for element string */

    printf("allocating %i for upe\n",(elend-URIew)+2);
    upe->name=malloc((elend-URIew)+2);
    if(!upe->name)
    {
      fprintf(stderr,
              "URIExtrude: Failed to malloc %i bytes for upe->name\n",
              (elend-URIew)+2);
      free(upe);
      return NULL;
    }

    index=0;

    //printf("URIew=%i elend=%i len=%i\npath(%i)=%s\n",URIew,elend,elend-URIew,(int)path,path);

    //while(URIew!=(elend+1))

    cpl=(elend+1)-URIew; /* cpl == number of characters to copy */

    while(cpl!=index)
    {
      printf("(URIew-(elend+1))=%i index=%i\n",(URIew-(elend+1)),index);

      if(*URIew!='/') /* don't keep any spurious /'s */
      {
        upe->name[index]=*URIew;
         ++URIew;
      } else {
        upe->name[index]='\0';
      }
      ++index;
    }
    upe->name[index]='\0';

    printf("upe->name=%s upe->next=NULL\n",upe->name);

    /* link it into the list */
    if(ue->path==NULL) {
      ue->path=upe;
      ue->path->next=NULL;
    } else {
      URIPathElement *upen;

      upen=ue->path;
      while(upen->next!=NULL)
      {
        //printf("[[%x %x  %s/]]\n", (int)upen, (int)upen->next, upen->name);
        upen=upen->next;
      }

      //printf("\n");

      upen->next=upe;
      upe->next=NULL;
    }

    //printf("{%c %x %x}\n",*URIew,URIew,URIewl);

    ++URIew; /* skip the / ready for next one */
  }

  //printf("exit URIExtrudepath\n");

  return URIew;
}

/*
 *  URIExtruded *URIExtrude(char *URI)
 *
 *  Take a URI and pull it out into a URIExtruded structure with
 *  URIPathElement structure.
 *
 */

URIExtruded *URIExtrude(char *URI)
{
  char *URIe=NULL, *URIew, *URIp;
  int rindex=0,windex=0,index=0;
  URIExtruded *ue;

  printf("strlen(URI)=%i\n",strlen(URI));

  /* malloc length of string+1 to decode any % sequences into */
  URIp=malloc(strlen(URI)+1);
  if(!URIp)
  {
    fprintf(stderr,"URIExtrude: Failed to malloc %i bytes for unescaped URI\n",
            (strlen(URI))+1);
    return NULL;
  }

  strcpy(URIp,URI);

  /* take the original URI and alter it, decoding all escape sequences */
  while(URIp[rindex]!='\0')
  {
    if(URIp[rindex]=='%')
    {
      if((dval(URIp[rindex+1])!=-1) && (dval(URIp[rindex+2])!=-1))
      {
        URIp[windex]=(dval(URIp[rindex+1])<<4)+dval(URIp[rindex+2]);
        rindex+=3;
        windex+=1;
        continue;
      } else {
        URIp[windex]=URIp[rindex];
        ++windex;
        ++rindex;
      }
    } else {
      if(rindex!=windex)
        URIp[windex]=URIp[rindex];
      ++windex;
      ++rindex;
    }
  }

  printf("windex=%i\n",windex);
  URIp[windex]='\0'; /* terminate */

  /* malloc 3x length of URI - all characters could need escaping */
  URIe=malloc((strlen(URIp)*3)+1);
  if(!URIe)
  {
    fprintf(stderr,"URIExtrude: Failed to malloc %i bytes for escaped URI\n",
            (strlen(URIp)*3)+1);
    return NULL;
  }

  /* escape any non standard characters in the URI */
  hex_escape_path_cpy(URIe, URIp);

  /* free up old unescaped copy */
  free(URIp);

  //printf("URIe=%s\n",URIe);

  /* before we start real processing make a work copy of URIe */
  URIew=URIe;

  /* and allocate space for URIExtruded structure */
  ue=malloc(sizeof(URIExtruded));
  if(!ue)
  {
    fprintf(stderr,"URIExtrude: Failed to malloc %i bytes for URIExtruded\n",
            sizeof(URIExtruded));
    free(URIe);
    return NULL;
  }

  memset(ue,0,sizeof(URIExtruded)); /* clear memory */

  /* http://user@www.moo.org:8080/directory1/directory2/index.html
   * ^^^^
   */

  if(strstr(URIew,"://"))
  {
    /* this is a literal url, we expect transport, host, maybe
     * user and port before a path */

    char *tranend=0;  /* http[:]//foo@www.monkey.com:8080/monkey/index.htm */
    char *pathstart=0;/* http://foo@www.monkey.com:8080[/]monkey/index.htm */
    char *userend=0;  /* http://foo[@]www.monkey.com:8080/monkey/index.htm */
    char *portstart=0;/* http://foo@www.monkey.com[:]8080/monkey/index.htm */
    char *hostend=0;  /* http://foo@www.monkey.com[:]8080/monkey/index.htm */
                      /* can differ from port start if there's no port */
#ifdef RESOLVENAMES
    struct hostent *hp;
    struct sockaddr_in address;
#endif

    tranend=strstr(URIew,"://");

    ue->transport=malloc((tranend-URIew)+2);
    if(!ue->transport)
    {
      fprintf(stderr,
              "URIExtrude: Failed to malloc %i bytes for ue->transport\n",
              (tranend-URIew)+1);
      free(URIe);
      return NULL;
    }

    index=0;

    while(URIew!=tranend)
    {
      ue->transport[index]=*URIew;
      ++URIew;
      ++index;
    }
    ue->transport[index]='\0';

    /* got the transport, skip the :// too */
    URIew+=3;

    /* find where the actual path starts, this will be the upper
     * limit for us finding @ and : for username/port
     */

    pathstart=strchr(URIew,'/');
    if(!pathstart)
      pathstart=URIew+strlen(URIew);

    /* find @ and : */
    if((strchr(URIew,'@')) < pathstart)
      userend=strchr(URIew,'@');

    if((strchr(URIew,':')) < pathstart)
      portstart=strchr(URIew,':');
    /* @ and : must be after each other */
    if((userend > portstart) && (portstart!=NULL))
    {

      fprintf(stderr,
              "URIExtrude: %s - @ and : do not follow each other\n",
              URIe);
      free(URIe);
      return NULL;
    }

    /* Now pull out user, a la :
     *
     * http://user@www.moo.org:8080/directory1/directory2/index.html
     *        ^^^^
     */

    if(userend)
    {
      ue->user=malloc((userend-URIew)+2);
      if(!ue->user)
      {
        fprintf(stderr,
                "URIExtrude: Failed to malloc %i bytes for ue->user\n",
                (userend-URIew)+2);
        free(URIe);
        return NULL;
      }

      index=0;

      while(URIew!=userend)
      {
        ue->user[index]=*URIew;
        ++URIew;
        ++index;
      }
      ue->user[index]='\0';

      ++URIew; /* skip the @ ready for next stage */
    }

    /* Now pull out the host :
     *
     * http://user@www.moo.org:8080/directory1/directory2/index.html
     *             ^^^^^^^^^^^
     */

    if(!portstart)
      hostend=pathstart;
    else
      hostend=portstart;

    index=0;

    ue->host=malloc((hostend-URIew)+2);
    if(!ue->host)
    {
      fprintf(stderr,
              "URIExtrude: Failed to malloc %i bytes for ue->host\n",
              (hostend-URIew)+2);
      free(URIe);
      return NULL;
    }

    while(URIew!=hostend)
    {
      ue->host[index]=*URIew;
      ++URIew;
      ++index;
    }
    ue->host[index]='\0';

#ifdef RESOLVENAMES

    /* we now have the host : attempt to resolve it
     *
     * what we do is to resolve the initial address back to an ip address,
     * and then do a reverse domain lookup to get the "proper" name of the
     * host.
     *
     * This way we don't end up indexing sites several times just because
     * they have several hostnames aliased to the same site.
     *
     */

    address.sin_family=AF_INET;

    /* is the hostname a dotted address? */
    if((address.sin_addr.s_addr=inet_addr(ue->host)) == (u_long)-1)
    {
      if ((hp = gethostbyname(ue->host)) == NULL) {
         fprintf(stderr,"URIExtrude: gethostbyname() failed for '%s'.\n",
                 ue->host);
         URIFreeExtruded(ue);
         free(URIe);
         return(0);
      }
    } else {
      if ((hp = gethostbyaddr((char *)&address.sin_addr,4,AF_INET)) == NULL) {
         fprintf(stderr,"URIExtrude: gethostbyaddr() failed for '%s'.\n",
                 ue->host);
         URIFreeExtruded(ue);
         free(URIe);
         return(0);
      }
    }
    printf("hp=0x%x\n");

    if(hp->h_name)
    {
      /* free what we had as the hostname */
      free(ue->host);
      ue->host=NULL;

      /* allocate and copy in what we found */
      ue->host=malloc(strlen(hp->h_name)+1);
      if(!ue->host)
      {
        fprintf(stderr,
                "URIExtrude: Failed to malloc %i bytes for ue->host (%s)\n",
                strlen(hp->h_name)+1,hp->h_name);
        URIFreeExtruded(ue);
        free(URIe);
        return NULL;
      }

      strcpy(ue->host,hp->h_name);
    }

#endif

    /* Deal with port, if there is one
     *
     * http://user@www.moo.org:8080/directory1/directory2/index.html
     *                         ^^^^
     */

    if(*URIew==':')
    {
      index=0;

      ++URIew; /* skip : */

      ue->port=malloc((pathstart-URIew)+2);
      if(!ue->port)
      {
        fprintf(stderr,
                "URIExtrude: Failed to malloc %i bytes for ue->port\n",
                (pathstart-URIew)+2);
        URIFreeExtruded(ue);
        free(URIe);
        return NULL;
      }

      while(URIew!=pathstart)
      {
        ue->port[index]=*URIew;
        ++URIew;
        ++index;
      }
      ue->port[index]='\0';
    }

    /* skip any leading / ready to process path */

    if(*URIew=='/')
      ++URIew;
  }


  URIew=URIExtrudePath(ue,URIew); /* turn path into linked list */

  if(!URIew) /* path extrude failed */
  {
    free(URIe);
    URIFreeExtruded(ue);
    return NULL;
  }

  if(ue->transport) printf("ue->transport = %s\n",ue->transport);
  if(ue->user     ) printf("ue->user      = %s\n",ue->user     );
  if(ue->host     ) printf("ue->host      = %s\n",ue->host     );
  if(ue->port     ) printf("ue->port      = %s\n",ue->port     );
  if(ue->leafname ) printf("ue->leafname  = %s\n",ue->leafname );

  {
    URIPathElement *upe;

    upe=ue->path;

    while(upe)
    {
      printf("%s/",upe->name);
      upe=upe->next;
    }

    printf(" - unextruded\n");
  }

  free(URIe);

  URIPack(ue);

  {
    URIPathElement *upe;

    upe=ue->path;

    while(upe)
    {
      printf("%s/",upe->name);
      upe=upe->next;
    }

    printf(" - unextruded\n");
  }

  return ue;
}

static void URIPathListDump(URIPathElement *upe)
{
  printf("%8s %8s  %s\n", "upe","upe->next","upe->name");

  while(upe)
  {
    printf("%8x %8x  %s\n",
           upe,upe->next,upe->name);

    upe=upe->next;
  }
}

/*  void URIPack(URIExtruded *URI)
 *
 *  Takes a URI and deals with any ..'s within it
 */

static void URIPack(URIExtruded *URI)
{

  URIPathElement *upe;

  URIPathListDump(URI->path);

  upe=URI->path;

  while(upe)
  {
    if(upe->next)
    {
      printf("upe=%x upe->next=%x\n",upe,upe->next);

      if(!strcmp(upe->next->name,".."))
      {
        URIPathElement *last,*temp;

        printf("{%i}\n",__LINE__);

        temp=upe->next->next;  /* store next pointer */
        free(upe->next->name); /* free the .. */
        printf("{%i}\n",__LINE__);
        free(upe->next);
        printf("{%i}\n",__LINE__);
        printf("free upe->next=%x\n",upe->next);
        upe->next=temp; /* fix list linkage */

        URIPathListDump(URI->path);

        printf("{%i}\n",__LINE__);

        /* find the list element that refers to _this_/.. */
        last=URI->path;
        printf("{%i}\n",__LINE__);
        while(last->next)
        {
          printf("%x,",last);
          if(last->next==upe)
            break;
          printf("{%i}\n",__LINE__);
          last=last->next;
        }
        printf("\n");

        printf("{%i}\n",__LINE__);

        /* if NULL referer must be URI->path itself */
        if(last->next==NULL)
          URI->path=upe->next;
        else
          last->next=upe->next;

        printf("{%i}\n",__LINE__);
        free(upe->name);
        printf("{%i}\n",__LINE__);
        //if(upe==URI->path) /* if we're freeing head of list fix URI->path */
          //URI->path=NULL;
        printf("{%i}\n",__LINE__);
        free(upe);
        printf("{%i}\n",__LINE__);

        upe=URI->path;
        if(!upe)
        {
          printf("break {%i} - upe = %x URI->path = %x\n",__LINE__,upe,URI->path);
          break;
        } else {
          continue;
        }

      }

      if(!upe) {
        printf("break {%i} - upe = %x URI->path = %x\n",__LINE__,upe,URI->path);
        break;
      } else {
        printf("break {%i} - upe = %x URI->path = %x\n",__LINE__,upe,URI->path);
        upe=upe->next;
      }
    } else {
      printf("break {%i} - upe = %x URI->path = %x\n",__LINE__,upe,URI->path);
      break;
    }
  }

  URIPathListDump(URI->path);

}

/*
 *  char *URIUnextrude(URIExtruded *URI)
 *
 *  Rebuild a text URI from a URIExtruded structure, the advantage
 *  of this being the URI will be canconicalised.
 *
 */

char *URIUnextrude(URIExtruded *URI)
{
  int length=1;
  URIPathElement *upe;
  char *out;

  if(!URI)
    return NULL;

  if(URI->transport)
  {
    length+=strlen(URI->transport)+3;  /* http:// */
    length+=strlen(URI->host)+1;
    if(URI->port)
      length+=strlen(URI->port)+1;
    if(URI->user)
      length+=strlen(URI->user)+1;
    if(URI->leafname)
      length+=strlen(URI->leafname)+1;
  }

  upe=URI->path;

  while(upe)
  {
    length+=strlen(upe->name)+1;;
    upe=upe->next;
  }

  //length+=128;

  out=malloc(length);

  if(out==NULL)
  {
    fprintf(stderr,"URIUnextrude: Failed to allocate %i bytes\n",length);
    return NULL;
  }

  if(URI->transport)
  {
    sprintf(out,"%s://",URI->transport);
    if(URI->user)
    {
      strcat(out,URI->user);
      strcat(out,"@");
    }
    strcat(out,URI->host);
    if(URI->port)
    {
      strcat(out,":");
      strcat(out,URI->port);
    }
    strcat(out,"/");
  }

  upe=URI->path;

  while(upe)
  {
    strcat(out,upe->name);
    strcat(out,"/");
    upe=upe->next;
  }

  if(URI->leafname)
    strcat(out,URI->leafname);

  return out;
}

/*  void URIFreeExtruded(URIExtruded *URI)
 *
 *  Free memory used by an extruded URI structure.
 */

void URIFreeExtruded(URIExtruded *URI)
{
  URIPathElement *pe=NULL;

  if(!URI)
    return;

  pe=URI->path;

  /* free URIPathElement linked list */
  while(pe!=NULL)
  {
    URIPathElement *tofree;
    printf("freeing %s\n",pe->name);
    free(pe->name);
    tofree=pe;
    pe=pe->next;
    printf("freeing %x (tofree)\n",(int)tofree); free(tofree);
    printf("pe now = %x\n",(int)pe);
  }

  if(URI->transport) free(URI->transport);
  if(URI->user)      free(URI->user);
  if(URI->host)      free(URI->host);
  if(URI->port)      free(URI->port);
  if(URI->leafname)  free(URI->leafname);

  /* finally free URIExtruded struct */
  printf("freeing %x (URI)\n",(int)URI); free(URI);
}

/*
 *  char *URIRelativeToLiteral(char *base, char *relative)
 *
 *  Turns a base and relative URI into a literal URI... ie :
 *
 *  base     = http://www.xargle.demon.co.uk/this/directory/here/
 *  relative = ../../papers/moo.html
 *  literal  = http://www.xargle.demon.co.uk/this/papers/moo.html
 *
 */

char *URIRelativeToLiteral(char *base, char *relative)
{
  URIExtruded *ue;
  URIPathElement *upe;
  char *r;

  printf("ue=URIExtrude(base);");
  ue=URIExtrude(base);
  printf("\n");

  if(!ue)
  {
    fprintf(stderr,"URIRelativeToLiteral: Failed to extrude base URL\n");
    return NULL;
  }

  //r=URIUnextrude(ue);
  //printf("URIUnextrude(ue)=%s\n",r);
  //free(r);

  r=URIExtrudePath(ue, relative);

  if(!r)
  {
    fprintf(stderr,"URIRelativeToLiteral: Failed to extrude relative path\n");
    return NULL;
  }

  URIPack(ue);

  //r=URIUnextrude(ue);
  //printf("%s\n",r);
  //free(r);

  r=URIUnextrude(ue);
  URIFreeExtruded(ue);

  return r;
}

/*
 *  char *URICanonicalise(char *URI)
 *
 *  Canconicalises (turns to base form) a URI. This gets rid of
 *  dodgy encodings, URI specs, host aliases etc, and weird
 *  path stuff.
 *
 */

char *URICanonicalise(char *URI)
{
  URIExtruded *ue;
  char *output;

  ue=URIExtrude(URI);
  if(!ue)
  {
    fprintf(stderr,"URICanonicalise: URIExtrude(%s) failed\n",URI);
    return NULL;
  }
  output=URIUnextrude(ue);
  if(!output)
  {
    fprintf(stderr,
            "URICanonicalise: URIUnextrude() failed of original URI %s\n",URI);
    URIFreeExtruded(ue);
    return NULL;
  }
  URIFreeExtruded(ue);

  return(output);
}
```

F.27 uri.h

```
/*
```

```
 * uri.h
 * Tony Howat, 1999/2000
 *
 * Routines for munging URIs, dealing with relative URIs etc.
 *
 * References RFC1808
 *
 */

/*
 *  These two structures are used to pull a URI apart ie :
 *
 *  http://user@www.moo.org:8080/directory1/directory2/index.html
 *  |      |    |          |                           |
 *  |      |    |          port                        leafname
 *  |      |    host
 *  |      username
 *  transport
 *
 *  Path section is turned into a linked list of URIPathElements.
 *
 */

typedef struct _URIPathElement
{
  char *name;                   /* element of path */
  struct _URIPathElement *next;
} URIPathElement;

typedef struct
{
  char *transport;
  char *user;
  char *host;
  char *port;
  URIPathElement *path;
  char *leafname;
} URIExtruded;

/*
 *  URIExtruded *URIExtrude(char *URI)
 *
 *  Take a URI and pull it out into a URIExtruded structure with
 *  URIPathElement structure.
 *
 */

URIExtruded *URIExtrude(char *URI);

/*
 *  char *URIUnextrude(char *URI)
 *
 *  Rebuild a text URI from a URIExtruded structure, the advantage
 *  of this being the URI will be canconicalised.
 *
 */

char *URIUnextrude(URIExtruded *URI);

/*  void URIFreeExtruded(URIExtruded *URI)
 *
 *  Free memory used by an extruded URI structure.
 */

void URIFreeExtruded(URIExtruded *URI);

/*
 *  char *URIRelativeToLiteral(char *base, char *relative)
 *
 *  Turns a base and relative URI into a literal URI... ie :
 *
 *  base     = http://www.xargle.demon.co.uk/this/directory/here/
 *  relative = ../../papers/moo.html
 *  literal  = http://www.xargle.demon.co.uk/this/papers/moo.html
 *
 */

char *URIRelativeToLiteral(char *base, char *relative);

/*
 *  char *URICanonicalise(char *URI)
 *
 *  Canconicalises (turns to base form) a URI. This gets rid of
 *  dodgy encodings, URI specs, host aliases etc, and weird
 *  path stuff.
 *
 *  Just calls URIExtrude, URIUnextrude and then URIFreeExtruded
 */

char *URICanonicalise(char *URI);
```

F.28 util.c

```
/*
 * util.c
 * Tony Howat, 1999/2000
 *
 * common utility functions
 *
 */

#include <stdio.h>
#include <string.h>
```

```
#include <stdlib.h>
#include "util.h"
#include <time.h>
#include <unistd.h>
#include <stdarg.h>

/*
 *  void *load_malloc(char *file, int *length, int *error)
 *
 *  Loads a file into memory, returning a pointer to a malloced block
 *  or null. File length is written to int pointed at by 2nd
 *  parameter. Any error is reported in int pointed to by 3rd
 *  parameter.
 *
 */

void *load_malloc(char *file, int *length, int *error)
{
   FILE *in;
   char *data;

   *error=0; *length=0;

   if((in=fopen(file,"r"))==NULL)
   {
     if(error)
       *error=uOPENFAIL;
     return NULL;
   }

   fseek(in,0,SEEK_END);
   *length=ftell(in);
   fseek(in,0,SEEK_SET);

   if((data=malloc((*length)+1))==NULL)
   {
     fclose(in);
     if(error)
       *error=uNOMEMORY;
     return NULL;
   }

   /* read the file */
   fread(data,1,*length,in);

   fclose(in);
   return(data);
}

/* Case insensitive strcmp */
int stricmp (char *s, char *t)
{
  for ( ; tolower(*s) == tolower(*t); ++s, ++t )
    if ( *s == '\0' )
      return(0);

  return (tolower(*s) - tolower(*t));
}

/* Case insensitive strncmp */
int strincmp (char *s, char *t, size_t n)
{
  while(n && (tolower(*s) == tolower(*t)))
  {
    if(*s=='\0')
      return(0);
    ++s; ++t; --n;
  }

  if(!n)
    return(0);
  else
    return (tolower(*s) - tolower(*t));
}

/* strip punctuation from leading and ending characters of a string */
char *strippunct(char *word)
{
  char *newword;
  int endindex;
  char *c;

  newword=word;

  /* we have a word, skip any punctuation at the start of it */
  while(*newword!='\0')
  {
    if(!ispunct(*newword))
      break;
    ++newword;
  }

  if(strlen(newword))
  {
    endindex=strlen(newword)-1;

    /* trim any punctuation at the end of it */

    while(endindex)
    {
      if(!ispunct(newword[endindex]))
        break;
      word[endindex]='\0';
      --endindex;
    }
  }
  /* now find any apostrophes and drop those
   */

  while((c=strstr(newword,"'")) != NULL) {
    printf("%s\n",newword);
    memmove(c,c+1,(strlen(c+1)+1)); /* shuffle it along */
  }

  return newword;
}

/* strip carriage returns from end of a string */
void stripcr(char *word)
{
  char *loc;

  loc=word+strlen(word)-1;

  while(loc!=word)
  {
    if(*loc=='\n')
      *loc='\0';
    else
      if(*loc=='\0')
        break;
    --loc;
  }
}

/* calculate the crc of a data block of length count */
int calccrc(char *ptr, int count)
{
   int crc, i;
   crc = 0;

   while(--count >= 0) {
     crc = crc ^ (int)*ptr++ << 8;
     for(i = 0; i < 8; ++i)
       if(crc & 0x8000)
         crc = crc << 1 ^ 0x1021;
       else
         crc = crc << 1;
     }
   return (crc);
}

/* Exponential hashing method
 *
 * Generates a hash based on
 * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
 * using int arithmetic, where s[i] is the
 * ith character of the string, n is the length of
 * the string, and ^ indicates exponentiation.
 */

int expohash(char *ptr, int count)
{
  int h=0, offset=0;

  while(offset!=count)
  {
    h=31*h+(*ptr++);
    offset++;
  }

  return h;
}

/* strlower - lower case a string */

void strlower(char *ptr)
{
  while (*ptr)
  {
    if(isupper(*ptr))
      *ptr=tolower(*ptr);
    ++ptr;
  }
}

/* function to replace newlines with spaces, and then
 * remove any double spaces to make optimal use of area
 * available - returns pointer to final character in string
 */
int makeoneline(char *input)
{
  char *c;

  c=input;

  /* replace newlines with spaces */
  while(*c!='\0')
  {
    if(*c=='\n')
      *c=' ';
    ++c;
  }

  while((c=strstr(input,"  ")) != NULL) {
    printf("%s\n",input);
    memmove(c,c+1,(strlen(c+1)+1)); /* shuffle it along */
  }

  return(strlen(input));
}

/* remove any leading spaces, return length */
```

```
int striplspace(char *input)
{
  if(*input==' ')
    memmove(input,input+1,(strlen(input+1)+1));

  return(strlen(input));
}

/* print a line to stdout, including pid and time for logging */
void log_event(char *fmt, ...)
{
  va_list argptr;
  char *cp;
  time_t t;

  time(&t);
  cp=ctime(&t);
  cp[strlen(cp)-1]='\0'; /* get rid of \n */
  printf("%i %s - ", getpid(), cp);
  va_start(argptr,fmt);
  vprintf(fmt,argptr);
  va_end(argptr);
  printf("\n");
}
```

F.29 util.h

```
/*
 * util.h
 * Tony Howat, 1999/2000
 *
 * common utility functions
 *
 */

#ifndef TRUE
  #define TRUE 1
#endif
#ifndef FALSE
  #define FALSE 0
#endif

#define uOPENFAIL 1  /* failure codes for load_malloc, can be returned */
#define uNOMEMORY 2  /* in error paramter                          */

/*
 *  void *load_malloc(char *file, int *length, int *error)
 *
 *  Loads a file into memory, returning a pointer to a malloced block
 *  or null. File length is written to int pointed at by 2nd
 *  parameter. Any error is reported in int pointed to by 3rd
 *  parameter.
 *
 */

void *load_malloc(char *file, int *length, int *error);

/* Case insensitive strcmp */
int stricmp (char *s, char *t);

/* Case insensitive strncmp */
int strincmp (char *s, char *t, size_t n);

/* strip punctuation from leading and ending characters of a string */
char *strippunct(char *word);

/* calculate the crc of a data block of length count */
int calccrc(char *ptr, int count);

/* Exponential hash
 *
 * Generates a hash based on
 * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
 * using int arithmetic, where s[i] is the
 * ith character of the string, n is the length of
 * the string, and ^ indicates exponentiation.
 */
int expohash(char *ptr, int count);

/* strlower - lower case a string */
void strlower(char *ptr);

/* strip carriage returns from end of a string */
void stripcr(char *word);

/* function to replace newlines with spaces, and then
 * remove any double spaces to make optimal use of area
 * available - returns pointer to final character in string
 */
int makeoneline(char *input);

/* remove any leading spaces, return length */
int striplspace(char *input);

/* print a line to stdout, including pid and time for logging */
void log_event(char *fmt, ...);
```